Properties, Learning Algorithms, and Applications of Chain Graphs and Bayesian Hypergraphs

Mohammad Ali Javidian

Follow this and additional works at: https://scholarcommons.sc.edu/etd

Part of the Computer Sciences Commons, and the Engineering Commons

Recommended Citation


This Open Access Dissertation is brought to you by Scholar Commons. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of Scholar Commons. For more information, please contact digres@mailbox.sc.edu.
Properties, Learning Algorithms, and Applications of Chain Graphs and Bayesian Hypergraphs

by

Mohammad Ali Javidian

Bachelor of Science
Shahid Bahonar University of Kerman 2003

Master of Science
Sharif University of Technology 2013

Submitted in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy in Computer Science and Engineering College of Engineering and Computing University of South Carolina 2019

Accepted by:

Marco Valtorta, Major Professor
Pooyan Jamshidi, Committee Member
Linyuan Lu, Committee Member
Chin-Tser Huang, Committee Member
John Rose, Committee Member

Cheryl L. Addy, Vice Provost and Dean of the Graduate School
DEDICATION

To Marco Valtorta, my dearest professor.
Probabilistic graphical models (PGMs) use graphs, either undirected, directed, or mixed, to represent possible dependencies among the variables of a multivariate probability distribution. PGMs, such as Bayesian networks and Markov networks, are now widely accepted as a powerful and mature framework for reasoning and decision making under uncertainty in knowledge-based systems. With the increase of their popularity, the range of graphical models being investigated and used has also expanded. Several types of graphs with different conditional independence interpretations - also known as Markov properties - have been proposed and used in graphical models.

The graphical structure of a Bayesian network has the form of a directed acyclic graph (DAG), which has the advantage of supporting an interpretation of the graph in terms of cause-effect relationships. However, a limitation is that only asymmetric relationships, such as cause and effect relationships, can be modeled between variables in a DAG. Chain graphs, which admit both directed and undirected edges, can be used to overcome this limitation. Today there exist three main different interpretations of chain graphs in the literature. These are the Lauritzen-Wermuth-Frydenberg, the Andersson-Madigan-Perlman, and the multivariate regression interpretations. In this thesis, we study these interpretations based on their separation criteria and the intuition behind their edges. Since structure learning is a critical component in constructing an intelligent system based on a chain graph model, we propose new feasible and efficient structure learning algorithms to learn chain graphs from data under the faithfulness assumption.

The proliferation of different PGMs that allow factorizations of different kinds leads us to consider a more general graphical structure in this thesis, namely directed acyclic
hypergraphs. Directed acyclic hypergraphs are the graphical structure of a new probabilistic graphical model that we call *Bayesian hypergraphs*. Since there are many more hypergraphs than DAGs, undirected graphs, chain graphs, and, indeed, other graph-based networks, Bayesian hypergraphs can model much finer factorizations and thus are more computationally efficient. Bayesian hypergraphs also allow a modeler to represent causal patterns of interaction such as Noisy-OR graphically (without additional annotations). We introduce global, local and pairwise Markov properties of Bayesian hypergraphs and prove under which conditions they are equivalent. We also extend the causal interpretation of LWF chain graphs to Bayesian hypergraphs and provide corresponding formulas and a graphical criterion for intervention.

The framework of graphical models, which provides algorithms for discovering and analyzing structure in complex distributions to describe them succinctly and extract unstructured information, allows them to be constructed and utilized effectively. Two of the most important applications of graphical models are causal inference and information extraction. To address these abilities of graphical models, we conduct a causal analysis, comparing the performance behavior of highly-configurable systems across environmental conditions (changing workload, hardware, and software versions), to explore when and how causal knowledge can be commonly exploited for performance analysis.
# Table of Contents

**DEDICATION** .......................... iii

**ABSTRACT** .......................... iv

**List of Tables** ........................ ix

**List of Figures** ........................ xi

**CHAPTER 1  Introduction** ........................ 1

**CHAPTER 2  LWF Chain Graphs** ........................ 8

  2.1 Basic Definitions and Concepts ........................ 9

  2.2 On the Properties of LWF Chain Graphs ........................ 11

  2.3 Finding Minimal Separators in LWF Chain Graphs ........................ 13

  2.4 Efficient Learning of LWF Chain Graphs under the Faithfulness Assumption 21

**CHAPTER 3  MVR Chain Graphs** ........................ 42

  3.1 Basic Definitions and Concepts ........................ 43

  3.2 On the Properties of MVR Chain Graphs ........................ 48

  3.3 Finding Minimal Separators in MVR Chain Graphs ........................ 65

  3.4 Order-Independent Structure Learning of Multivariate Regression Chain Graphs ........................ 74
3.5 A Decomposition-Based Algorithm for Structure Learning of MVR Chain Graphs .................................................. 93
3.6 Discussion and Conclusion ...................................................... 117

CHAPTER 4 AMP CHAIN GRAPHS ................................................. 120
4.1 Basic Definitions and Concepts ................................................. 121
4.2 On the Properties of AMP Chain Graphs .................................. 124
4.3 Finding Minimal Separators in AMP Chain Graphs ...................... 126
4.4 Order-Independent Structure Learning of AMP Chain Graphs ........ 134
4.5 A Decomposition-Based Algorithm for Learning the Structure of AMP CGs 144
4.6 Discussion and Conclusion ...................................................... 161

CHAPTER 5 BAYESIAN HYPERGRAPHS ....................................... 166
5.1 Basic Definitions and Concepts ................................................. 168
5.2 Markov properties for undirected graphs .................................... 177
5.3 Bayesian Hypergraphs .......................................................... 179
5.4 Intervention in Bayesian hypergraphs ........................................ 193

CHAPTER 6 CAUSAL TRANSFER LEARNING ............................... 205
6.1 Introduction ................................................................. 205
6.2 Causal Graphs ............................................................ 208
6.3 Research Questions and Methodology ..................................... 210
6.4 Identification of Causal Effects (RQ1) ...................................... 212
6.5 Transportability of Causal and Statistical Relations Across Environments (RQ2) .................................................. 215
6.6 Generalizing Statistical Findings Across Sampling Conditions (RQ3) . . . . 217
6.7 Threats to Validity ................................................................. 218

CHAPTER 7 CONCLUDING REMARKS ........................................... 220

BIBLIOGRAPHY ................................................................. 225

APPENDIX A PROOFS OF CORRECTNESS OF THEOREMS AND ALGORITHMS IN SECTION 2.4 ................................................................. 239

APPENDIX B PROOF OF THEOREM 3.10 ........................................ 243

APPENDIX C PROOFS OF THEORETICAL RESULTS IN SECTION 3.5 ........ 247

APPENDIX D PROOFS OF CORRECTNESS OF ALGORITHMS 11 AND 12 .............................. 253

APPENDIX E PROOFS OF CORRECTNESS OF THE ALGORITHMS IN SECTIONS 4.4 AND 4.5 . 255
# List of Tables

Table 2.1 The trace table of Algorithm 4 for $i = 3$ and order$_{1}(V) = (d, e, a, c, b)$. . . 29

Table 2.2 The trace table of Algorithm 4 for $i = 3$ and order$_{2}(V) = (d, c, e, a, b)$. . . 29

Table 2.3 Results for discrete samples from the ASIA, INSURANCE, ALARM, and HAILFINDER networks respectively. Each row corresponds to the significance level: $\alpha = 0.05/0.01/0.005$ respectively. . . . . . . . . . 41

Table 3.1 The trace table of Algorithm 9 for $i = 3$ and order$_{1}(V) = (d, e, a, c, b)$. . . 79

Table 3.2 The trace table of Algorithm 9 for $i = 3$ and order$_{2}(V) = (d, c, e, a, b)$. . . 79

Table 3.3 Order-dependence issues and corresponding modifications of the PC-like algorithm that remove the problem. “Yes” indicates that the corresponding aspect of the graph is estimated order-independently in the sample version. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 88

Table 3.4 Results for discrete samples from the ASIA network. Each row corresponds to the significance level: $\alpha = 0.05/0.01/0.005$ respectively. . . . 115

Table 3.5 Results for discrete samples from the INSURANCE network. Each row corresponds to the significance level: $\alpha = 0.05/0.01/0.005$ respectively. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 116

Table 3.6 Results for discrete samples from the ALARM network. Each row corresponds to the significance level: $\alpha = 0.05/0.01/0.005$ respectively. . 116

Table 3.7 Results for discrete samples from the HAILFINDER network. Each row corresponds to the significance level: $\alpha = 0.05/0.01/0.005$ respectively. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 117

Table 4.1 The trace table of Algorithm 17 for $i = 1$ and order$_{1}(V) = (d, c, b, a, e)$. For simplicity, we define $ADJ_H(u) := [ad_H(u) \cup ad_H(ad_H(u))] \setminus \{u, v\}$. . . 138

Table 4.2 The trace table of Algorithm 17 for $i = 1$ and order$_{2}(V) = (d, e, a, c, b)$. For simplicity, we define $ADJ_H(u) := [ad_H(u) \cup ad_H(ad_H(u))] \setminus \{u, v\}$. . . 138
Table 4.3 The trace table of Algorithm 18 for $i = 1$, order$_1(V) = (d, c, b, a, e)$, and order$_2(V) = (d, e, a, c, b)$. For simplicity, we define $\text{ADJ}_H(u) := \{\text{ad}_H(u) \cup \text{ad}_H(\text{ad}_H(u))\} \setminus \{u, v\}$. ................................. 142

Table 4.4 Order-dependence issues and corresponding modifications of the PC-like algorithm that remove the problem. “Yes” indicates that the corresponding aspect of the graph is estimated order-independently in the sample version. ................................. 144

Table 4.5 Results for discrete samples from the ASIA, INSURANCE, ALARM, and HAILFINDER networks respectively. Each row corresponds to the significance level: $\alpha = 0.05/0.01/0.005$ respectively. ................................. 162

Table 5.1 The conditional probability distribution $P(C|A, B)$ for a model with: (a) noisy functional dependence (Noisy-OR), and therefore $P(C = n|A = y, B = y) = P(C = n|A = y)P(C = n|B = y)$; (b) non-noisy functional dependence. ................................. 167

Table 5.2 Factorizations and corresponding BH representations ................................. 185

Table 6.1 Overview of the real-world subject systems ................................. 213

Table 7.1 Properties of chain graphs under different interpretations ................................. 223

Table 7.2 List of publications for each chapter ................................. 224
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Test for minimal separation in LWF CGs</td>
<td>16</td>
</tr>
<tr>
<td>2.2</td>
<td>Finding restricted separation set in LWF CGs</td>
<td>19</td>
</tr>
<tr>
<td>2.3</td>
<td>The procedure of learning the structure of an LWF chain graph from a faithful distribution</td>
<td>21</td>
</tr>
<tr>
<td>2.4</td>
<td>(a) The DAG $G$, (b) the skeleton returned by Algorithm 4 with order1$(V)$, (c) the skeleton returned by Algorithm 4 with order2$(V)$</td>
<td>28</td>
</tr>
<tr>
<td>2.5</td>
<td>(a) The DAG $G$, (b) the CG returned after the complex recovery phase of Algorithm 4 with order1$(V)$, (c) the CG returned after the complex recovery phase of Algorithm 4 with order3$(V)$</td>
<td>30</td>
</tr>
<tr>
<td>2.6</td>
<td>Performance of the original and stable LCD and PC-like algorithms for randomly generated Gaussian chain graph models: average over 30 repetitions with 50 variables, expected degree $N = 3$, and significance levels $\alpha = 0.05, 0.005$</td>
<td>38</td>
</tr>
<tr>
<td>2.7</td>
<td>Performance of the original and stable LCD and PC-like algorithms for randomly generated Gaussian chain graph models: average over 30 repetitions with 100 variables, expected degree $N = 3$, and significance levels $\alpha = 0.05, 0.005$</td>
<td>39</td>
</tr>
<tr>
<td>3.1</td>
<td>(Lauritzen and Richardson, 2002) A nonmaximal ancestral graph</td>
<td>46</td>
</tr>
<tr>
<td>3.2</td>
<td>An MVR CG with chain components: $\mathcal{T} = {T_1 = {a, b}, T_2 = {c, d}, T_3 = {e, f}, T_4 = {g, h}}$</td>
<td>47</td>
</tr>
<tr>
<td>3.3</td>
<td>(a) A directed graph including a vertex $H$ for an unobserved variable, (b) the independence structure encoded by the MVR CG</td>
<td>49</td>
</tr>
<tr>
<td>3.4</td>
<td>A gene and disease example with MVR CG representation, BN representation and MN representation (Sonntag and Peña, 2015b)</td>
<td>50</td>
</tr>
<tr>
<td>3.5</td>
<td>an MVR CG with chain components: $\mathcal{T} = {T_1 = {1, 2, 3, 4}, T_2 = {5, 6}, T_3 = {7}}$</td>
<td>55</td>
</tr>
</tbody>
</table>
Figure 3.6 An MVR chain graph with chain components: $\mathcal{T} = \{a, b, c\}, \{d, e, f\}$. 59

Figure 3.7 Finding minimal separators in a maximal ancestral graph. 69

Figure 3.8 Finding restricted separator in a maximal ancestral graph. 72

Figure 3.9 The Rules (Sonntag and Peña, 2012) 75

Figure 3.10 (a) The DAG $G$, (b) the skeleton returned by Algorithm 9 with order$_1(V)$, (c) the skeleton returned by Algorithm 9 with order$_2(V)$. 78

Figure 3.11 (a) The DAG $G$, (b) the CG returned after the $\nu$-structure recovery phase of Algorithm 9 with order$_1(V)$, (c) the CG returned after the $\nu$-structure recovery phase of Algorithm 9 with order$_3(V)$. 80

Figure 3.12 Possible mixed graph after $\nu$-structure recovery phase of the sample version of the PC-like algorithm. 80

Figure 3.13 The first two rows show the performance of the original (OPC) and stable PC-like (SPC) algorithms for randomly generated Gaussian chain graph models: average over 30 repetitions with 50 variables correspond to $N = 2$, and the significance level $\alpha = 0.001$. The last two rows show the performance of the original (OPC) and stable PC-like (SPC) algorithms for randomly generated Gaussian chain graph models: average over 30 repetitions with 1000 variables correspond to $N = 2$, sample size $S=50$, and the significance level $\alpha = 0.05, 0.01, 0.005, 0.001$. 92

Figure 3.14 The procedure of learning the structure of an essential MVR chain graph from a faithful distribution. 93

Figure 3.15 (a) An MVR CG $G$. (b) The augmented graph $G'$, which is also a triangulated graph $G''$. 96

Figure 3.16 An $m$-separation tree. 97

Figure 3.17 Construction the $m$-separation tree. (a) An MVR CG. (b) Domain knowledge of associations. (c) The undirected graph and triangulation. (d) The $m$-separation tree $T$. 102
Figure 3.18 Insufficiency of having a hypergraph that contains both \( u \) and its parent set for every \( u \in V \). (a) An MVR CG. (b) Domain knowledge of associations. (c) The undirected graph constructed by union of complete graphs corresponding to each hyperedge, which is also a triangulated graph. (d) The junction tree \( T \). (e) Local skeleton for every node of \( T \). (f) The global skeleton and all \( v \)-structures.

Figure 3.19 Insufficiency of having a hypergraph that contains both \( u \) and its boundary set for every \( u \in V \). (a) An MVR CG. (b) Domain knowledge of associations. (c) The undirected graph constructed by union of complete graphs corresponding to each hyperedge, which is also a triangulated graph. (d) The junction tree \( T \), which is not an \( m \)-separation tree.

Figure 3.20 (a) Local skeletons for every node of \( T \). (b) The global skeleton and all \( v \)-structures.

Figure 3.21 The Rules (Sonntag and Peña, 2012)

Figure 3.22 Error measures of the decomposition-based and PC-like algorithms for randomly generated Gaussian chain graph models: average over 25 repetitions with 30 variables. The four rows correspond to \( N = 2 \) and 8. The three columns give three error measures: TPR, FPR and ACC in each setting respectively. In each plot, the solid (blue)/dashed (green)/dotted (red) lines correspond to significance levels \( \alpha = 0.05/0.01/0.005 \).

Figure 3.23 Error measure SHD of the decomposition-based and PC-like algorithms for randomly generated Gaussian chain graph models: average over 25 repetitions with 30 variables. The first row correspond to \( N = 2 \), the second row correspond to \( N = 5 \), and the third row correspond to \( N = 8 \). The first two columns correspond to the essential recovery while the last two columns correspond to the minimum bidirected recovery respectively. In each plot, the solid (blue)/dashed (green)/dotted (red) lines correspond to significance levels \( \alpha = 0.05/0.01/0.005 \).

Figure 3.24 Running times of the decomposition-based and PC-like algorithms for randomly generated Gaussian chain graph models: average over 25 repetitions with 30 variables correspond to \( N = 2 \). The first two columns correspond to the essential recovery algorithm while the last two columns correspond to the minimum bidirected recovery respectively. In each plot, the solid (blue)/dashed (green)/dotted (red) lines correspond to significance levels \( \alpha = 0.05/0.01/0.005 \).
Figure 4.1

(a) Triplexes and (b) the corresponding augmented triplex, (c) the
four configurations that define the bi-flag; (d) the corresponding
augmented bi-flag. The “?" indicates that either X − Y ∈ G, X →
Y ∈ G, Y → X ∈ G, or X and Y are not adjacent in G. . . . . . . . . . . 123

Figure 4.2

(a) The AMP CG G, (b) An(X ∪ Y ∪ A), (c) the undirected edges in
Co(An(X ∪ Y ∪ A)), (d) G[X ∪ Y ∪ A], and (e) (G[X ∪ Y ∪ A])a . . . . . 125

Figure 4.3

Finding a minimal separator in an AMP chain graph.

Figure 4.4

Finding restricted separator in an AMP chain graph. . . . . . . . . . . . 132

Figure 4.5

Rules R1-R4 (Peña, 2012) . . . . . . . . . . . . . . . . . . . . . . . . . 136

Figure 4.6

(a) The DAG G, (b) the skeleton returned by Algorithm 17 with
order1 (V), (c) the skeleton returned by Algorithm 17 with order2 (V). . . 137

Figure 4.7

(a) The DAG G, (b) the CG returned by Algorithm 17 with order1 (V),
(c) the CG returned by Algorithm 17 with order3 (V). . . . . . . . . . . 139

Figure 4.8

The procedure for learning the structure of the largest deflagged
AMP CG from a faithful distribution. . . . . . . . . . . . . . . . . . . . 145

Figure 4.9

(a) An AMP CG G. (b) The augmented graph Ga , which is also an
undirected independence graph. (c) The triangulated graph (Ga )t . . . . . 146

. . . . . . . . . . 130

Figure 4.10 The p-separation tree of CG G in Figure 4.9 . . . . . . . . . . . . . . . 147
Figure 4.11 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 158
Figure 4.12 The first two columns show the running times of the decomposition-based and
PC-like algorithms for randomly generated Gaussian chain graph models: average over 30 repetitions with 50 variables correspond to N = 2,3 and significance
levels α = 0.005. In each plot, the solid blue line corresponds to the LCD algorithm, the dashed red line corresponds to the original PC-like algorithm, and the
dotted grey line corresponds to the stable PC-like algorithm. The third column
shows the running times of the decomposition based (LCD) algorithm for randomly generated Gaussian chain graph models: average over 30 repetitions with
50 variables correspond to N = 2, and the significance level α = 0.05, 0.01, 0.005.
In each plot, the solid blue line corresponds to the α = 0.05, the dashed red line
corresponds to α = 0.01, and the dotted grey line corresponds to α = 0.005. . .

xiv

. . . 159


Figure 4.13  The first two columns show the number of independence tests used by the decomposition-based and PC-like algorithms for randomly generated Gaussian chain graph models: average over 30 repetitions with 50 variables corresponding to average degrees $N = 2, 3$ and significance level $\alpha = 0.005$. In each plot, the solid blue line corresponds to the LCD algorithm, the dashed red line corresponds to the original PC-like algorithm, and the dotted grey line corresponds to the stable PC-like algorithm. The third column shows the number of independence tests used by the decomposition based (LCD) algorithm for randomly generated Gaussian chain graph models: average over 30 repetitions with 50 variables corresponding to average degree $N = 2$, and significance levels $\alpha = 0.05, 0.01, 0.005$. In each plot, the solid blue line corresponds to $\alpha = 0.05$, the dashed red line corresponds to $\alpha = 0.01$, and the dotted grey line corresponds to $\alpha = 0.005$.

Figure 4.14  Original ASIA network; networks learned using LCD and PC-like algorithms, respectively.

Figure 5.1  A simple DAG $G$.

Figure 5.2  (1) a DAH $\mathcal{H}$. (2) the canonical DAG of $\mathcal{H}$. (3) the shadow of $\mathcal{H}$.

Figure 5.3  (1) a simple chain graph $G$; (2) the corresponding DAH of $G$ in the LWF interpretation.

Figure 5.4  Relationship between chain graphs and directed acyclic hypergraphs.

Figure 5.5  A simple Bayesian hypergraph $\mathcal{H}$.

Figure 5.6  (1) a chain graph $G$; (2) a Bayesian hypergraph $\mathcal{H}$.

Figure 5.7  A model of heart disease.

Figure 5.8  An example of Noisy-OR: obesity.

Figure 5.9  An example of Noisy-OR: Atheriosclerosis.

Figure 5.10  Two Bayesian hypergraphs $\mathcal{H}_1$ (left), $\mathcal{H}_2$ (right) with the same global Markov properties but different forms of factorizations.

Figure 5.11  (1) a simple chain graph $G$; (2) The moral graph $G^m$ of $G$; (3) $\mathcal{M}(\mathcal{H})$ where $\mathcal{H}$ is the canonical LWF DAH of $G$.

Figure 5.12  (a) A chain graph $G$; (b) The graph $\hat{G}$ obtained from $G$ through the redirection procedure; (c) The graph $G_0$ obtained from $G$ by deleting variables in $A$. 

xv
Figure 5.13 (a) A chain graph $G$; (b) the canonical LWF DAH $\mathcal{H}$ of $G$; (c) the resulting hypergraph $\hat{\mathcal{H}}$ after performing the graphical procedure on $\mathcal{H}$ when the variable $c$ is intervened upon.

Figure 5.14 Commutative diagram of factorization equivalence.

Figure 6.1 Exploiting causal inference for performance analysis.

Figure 6.2 Causal graph for x264 deployed on internal server Feature 1 and used version 2.76.2 of x264 and used a small video for encoding. For all figures we do not show options that do not affect on performance.

Figure 6.3 Causal graph for XGBoost12 with CNAE-9 data set, deployed on Feature 4. Performance nodes are: train-time, test-time, and accuracy.

Figure 6.4 Selection diagram for SPEAR in two environments: one with measured solving time, deployed on a private server, version 2.7, SAT size 10286, and another deployed on Azure Cloud.

Figure 6.5 Selection diagram for XGBoost12 deployed on two environments: one deployed on a private server Feature 4, with covtype dataset, and another with the same characteristics but deployed on Azure Cloud. Performance nodes are: train-time, test-time, and accuracy.

Figure 6.6 The causal graph $G_s$ for SQLite in the environment with Feature 20 and version 3.7.6.3.

Figure A.1 (a) The LWF CG $G$, (b) the skeleton of $G$, (c) $H^*$ before executing line 19 in Algorithm 4, and (d) $H^*$ after executing line 19 in Algorithm 4.

Figure E.1 (a) AMP CG $G$, (b) augmented graph $G^a$, (c) triangulated graph $(G^a)'$, and (d) $p$-separation tree $T$. 

xvi
Chapter 1

Introduction

Probabilistic graphical models (PGMs), and their use for reasoning intelligently under uncertainty, emerged in the 1980s within the statistics and artificial intelligence communities (Pearl, 1988; Neapolitan, 1990). Probabilistic graphical models are now widely accepted as a powerful and mature tools for reasoning under uncertainty. Unlike some of the ad hoc approaches taken in early experts systems, PGMs are based on the strong mathematical foundations of graph and probability theory. In fact, any PGM consists of two main components: (1) a graph that defines the structure of that model; and (2) a joint distribution over random variables of the model. They can be used for a wide range of reasoning tasks including prediction, monitoring, diagnosis, risk assessment and decision making (Spirtes, Glymour, and Scheines, 2000; Xiang, 2002; Jensen and Nielsen, 2007; Fenton and Neil, 2018). The main advantages of using PGMs compared to other models are that the representation is intuitive, inference can be done efficiently and efficient learning algorithms exist. This has led PGMs to arguably become the most important architecture for reasoning with uncertainty in artificial intelligence (Koller and Friedman, 2009; Neapolitan and Jiang, 2018). There are many efficient algorithms for both inference and learning available in open-source (e.g., (Højsgaard, Edwards, and Lauritzen, 2012; Nagarajan, Scutari, and Lèbre, 2013; Scutari and Denis, 2015)) and commercial software (e.g., Hugin, Netica, GeNiE, and BayesiaLab). Moreover, their power and efficacy has been proven through their successful application to an enormous range of real-world problem domains.

One of the most basic subclasses of PGMs is Markov networks. The graphical framework of Markov networks are undirected graphs (UGs), in which each undirected edge
represents a direct correlation between the two variables it connects, while no edge means that the variables are not directly correlated. The best known and most widely used PGM class, however, is Bayesian networks. The graphical structures of Bayesian networks are directed acyclic graphs (DAGs). In a DAG the directed edges can be seen as representing cause and effect relationships.

Despite the fact that PGMs, such as Bayesian networks and Markov networks, have been largely developed and used in real world applications, there is an increasing tendency by researchers to go beyond Bayesian networks and Markov networks. For example, systems containing both causal and non-causal relationships are mostly modeled with directed acyclic graphs. An alternative approach is using chain graphs (CGs). Chain graphs may have both directed and undirected edges under the constraint that there do not exist any semi-directed (partially directed) cycles (Drton, 2009). So, CGs may contain two types of edges, the directed type that corresponds to the causal relationship in DAGs and a second type of edge representing a symmetric relationship (Sonntag, 2016). In particular, \( X_1 \) is a direct cause of \( X_2 \) only if \( X_1 \to X_2 \) (i.e., \( X_1 \) is a parent of \( X_2 \)), and \( X_1 \) is a (possibly indirect) cause of \( X_2 \) only if there is a directed path from \( X_1 \) to \( X_2 \) (i.e., \( X_1 \) is an ancestor of \( X_2 \)). So, while the interpretation of the directed edge in a CG is quite clear, the second type of edge can represent different types of relations and, depending on how we interpret it in the graph, we say that we have different CG interpretations with different separation criteria, i.e. different ways of reading conditional independences from the graph, and different intuitive meaning behind their edges. The three following interpretations are the best known in the literature. The first interpretation (LWF) was introduced by Lauritzen, Wermuth and Frydenberg (Lauritzen and Wermuth, 1989; Frydenberg, 1990) to combine DAGs and undirected graphs (UGs). The second interpretation (AMP), was introduced by Andersson, Madigan and Perlman, and also combines DAGs and UGs but with a separation criterion that more closely resembles the one of DAGs (Andersson, Madigan, and Perlman, 1996). The third interpretation, the multivariate regression interpretation (MVR), was introduced
by Cox and Wermuth (Cox and Wermuth, 1993, 1996) to combine DAGs and bidirected (covariance) graphs.

**Problem 1. Properties of Chain Graphs (Chapters 2-4).** The different chain graph interpretations have been studied independently and over time different conditional independence interpretations - also known as Markov properties - have been proposed for each of them. This has however led to confusion regarding what Markov properties exist for what interpretation and under which conditions they are equivalent, especially in the case of MVR chain graphs. In this thesis we do therefore review some of fundamental concepts (e.g., Markov properties and unique representation of those chain graphs that induce the same conditional independence restrictions in a certain CG interpretation) and study how they are defined for different CG interpretations to give a coherent overview of the research performed. Also, we address the problem of finding a minimal separator in a chain graph, namely, finding a set $Z$ of nodes that separates a given non-adjacent pair of nodes such that no proper subset of $Z$ separates that pair. We analyze several versions of this problem and offer polynomial time algorithms for each. These include finding a minimal separator from a restricted set of nodes, finding a minimal separator for two given disjoint sets, and testing whether a given separator is minimal.

**Problem 2. Learning Chain Graphs (Chapters 2-4).** One important aspect of PGMs in general, and chain graphs especially, is the possibility of learning the structure of models directly from sampled data. Four *constraint-based* learning algorithms, that use a statistical analysis to test the presence of a conditional independency, exist for learning CGs: the inductive causation like (IC-like)/PC-like algorithms (Studený, 1997; Peña and Gómez-Olmedo, 2016; Sonntag and Peña, 2012), the answer set programming (ASP) algorithms (Peña, 2018b; Sonntag et al., 2015a), the decomposition-based algorithm called LCD (Ma, Xie, and Geng, 2008) and the inclusion optimal (CKES) algorithm (Peña, Sonntag, and Nielsen, 2014). The former two have implementations for all three CG interpretations, while the latter two are only available for LWF CGs (Sonntag, 2016). In this disserta-
tion, we present an order-independent PC-like algorithm (a modified version of the IC-like algorithm proposed in (Studený, 1997)) for learning the structure of all three different interpretations of chain graphs and two decomposition-based algorithms, one for AMP CGs and another for MVR CGs.

Problem 3. Bayesian Hypergraphs (Chapter 5). A PGM can be seen as a factorization of a joint probability distribution of the state of a system. Factorizing a large joint probability distribution has many benefits (Sonntag, 2016), e.g.: (1) it illuminates the conditional independences between the variables in the distribution, and (2) instead of having one large joint probability distribution we get multiple smaller probability distributions. This allows for efficient use of space since the size of a joint probability distribution grows exponentially with the number of nodes while the total size of local probability distributions only grows quasi-linearly if most variables are conditionally independent. Multiple small probability distributions generally also allow us to do calculations faster than using a single joint probability distribution. The factorization of a joint distribution not only is central for representation but also plays a key role in making inference feasible (Koller and Friedman, 2009). The proliferation of different PGMs that allow factorizations of different kinds leads us to consider a more general graphical structure. For this purpose, we propose a directed acyclic hypergraph framework for a probabilistic graphical model that we call Bayesian hypergraphs. The space of directed acyclic hypergraphs is much larger than the space of chain graphs. Hence Bayesian hypergraphs can model much finer factorizations than Bayesian networks or LWF chain graphs and provide simpler and more computationally efficient procedures for factorization and intervention.

Problem 4. Causal Transfer Learning (Chapter 6). The framework of graphical models, which provides algorithms for discovering and analyzing structure in complex distributions to describe them succinctly and extract the unstructured information, allows them to be constructed and utilized effectively. Causal inference and information extraction are two of the most important applications of graphical models. To demonstrate the abilities
of graphical models in these fields, we conduct a causal analysis, comparing performance behavior of highly-configurable systems across environmental conditions (changing work-load, hardware, and software versions), to explore when and how causal knowledge can be commonly exploited for performance analysis.

We list below the specific contributions achieved in this thesis by chapter and include the publications in which they were presented:

- Chapters 2-4, properties of chain graphs (Javidian and Valtorta, 2018c,b; Javidian, Valtorta, and Jamshidi, 2019a):
  1. introduce an alternative local Markov property for MVR chain graphs, which is equivalent to other Markov properties in the literature for compositional semi-graphoids.
  2. show a comparison of different proposed Markov properties for MVR chain graphs in the literature and conditions under which they are equivalent.
  3. propose an alternative explicit factorization criterion for MVR chain graphs based on the proposed factorization criterion for acyclic directed mixed graphs.
  4. address the problem of finding a minimal separator in (AMP, LWF, and MVR) chain graphs, namely, finding a set $Z$ of nodes that separates a given non-adjacent pair of nodes such that no proper subset of $Z$ separates that pair. Several versions of this problem are analyzed and polynomial time algorithms are offered for each. These include finding a minimal separator from a restricted set of nodes, finding a minimal separator for two given disjoint sets, and testing whether a given separator is minimal.

- Chapters 2-4, learning chain graphs (Javidian and Valtorta, 2019a; Javidian, Valtorta, and Jamshidi, 2019b,a):
1. propose an order-independent PC-like algorithm for learning the structure of all three different interpretations of chain graphs under the faithfulness assumption;

2. propose a decomposition approach for recovering structures of MVR CGs;

3. propose a decomposition approach for recovering structures of AMP CGs;

4. experimentally compare the performance of proposed algorithms with other existing algorithms for each interpretation of CGs, and show that the proposed algorithms are comparable (or superior) to the algorithms in the literature in terms of error measures and runtime;

5. provide data and an R package that implements the proposed algorithms.

- Chapter 5, Bayesian hypergraphs (Javidian et al., 2018; Wang et al., 2019):
  1. proposes a directed acyclic hypergraph framework for a probabilistic graphical model that is called Bayesian hypergraphs;
  2. proves that Bayesian hypergraphs can model much finer factorizations than Bayesian networks or LWF chain graphs;
  3. explains the power of Bayesian hypergraphs to represent causal patterns of interaction such as Noisy-OR graphically (without additional annotations);
  4. introduces global, local and pairwise Markov properties of Bayesian hypergraphs and proves under which conditions they are equivalent;
  5. defines a projection operator, called shadow, that maps Bayesian hypergraphs to chain graphs, and shows that the Markov properties of a Bayesian hypergraph are equivalent to those of its corresponding chain graph;
  6. extends the causal interpretation of LWF chain graphs to Bayesian hypergraphs and provides corresponding formulas and a graphical criterion for intervention.

- Chapter 6, causal transfer learning (Javidian, Jamshidi, and Valtorta, 2019):
1. conducts a causal analysis, comparing performance behavior of highly configurable systems across environmental conditions (changing workload, hardware, and software versions), to explore when and how causal knowledge can be commonly exploited for performance analysis;

2. shows the possibility of identifiability of causal effects of configuration options on performance from observational studies alone;

3. shows that many of causal or statistical relations about performance behavior can be transferred across environments even in the most severe changes we explored, and that transportability is actually trivial for many environmental changes;

4. indicates the recoverability of conditional probabilities from selection-biased data to the entire population.
Chapter 2

LWF Chain Graphs

*LWF Chain graphs* were introduced by Lauritzen, Wermuth and Frydenberg (Frydenberg, 1990; Lauritzen and Wermuth, 1989) as a generalization of graphical models based on undirected graphs and directed acyclic graphs and widely studied in (Lauritzen, 1996; Richardson, 1998; Lauritzen and Richardson, 2002; Cowell et al., 1999; Drton, 2009; Ma, Xie, and Geng, 2008; Peña, Sonntag, and Nielsen, 2014; Peña, 2015; Sonntag and Peña, 2015a; Sonntag, 2014; Studeny, 1997; Volf and Studeny, 1999; Studeny, 2005; Studeny, Roverato, and Š. Štěpánová, 2009; Roverato, 2005; Roverato and Rocca, 2006; Roverato, 2017).

In this chapter, we address the problem of finding minimal separators in LWF chain graphs, because minimality is a desirable property to ensure efficiency and usability. Moreover, finding minimal separators is useful for learning and inference tasks (Acid and Campos, 1996; Tian, Paz, and Pearl, 1998).

As we mentioned in Chapter 1, the possibility of learning the structure of graphical models directly from sampled data is an important aspect of PGMs. So, in this chapter, we propose a PC-like algorithm for learning the structure of LWF chain graphs under the faithfulness assumption and we show that our proposed approach is comparable to the existing methods in terms of error measures and runtime.

The chapter is organized as follows. In section 2.1, we define the notation and terminology used throughout the chapter. In section 2.2, we take a closer look at LWF CGs in terms of separation criterion, Markov properties, factorization, Markov equivalence class, and largest chain graphs. In section 2.3, we propose and solve an optimization problem related to the separation in LWF chain graphs. In section 2.4, We present a PC-like al-
algorithm that finds the structure of chain graphs under the faithfulness assumption. We show that our PC-like algorithm is order dependent, in the sense that the output can depend on the order in which the variables are given. This order dependence can be very pronounced in high-dimensional settings. We propose two modifications of the PC-like algorithm that remove part or all of this order dependence. Simulation results under a variety of settings demonstrate the competitive performance of the modified PC-like algorithms in comparison with the original PC-like algorithm in low-dimensional settings and improved performance in high-dimensional settings. We show that our approach is comparable to the decomposition-based method proposed in (Ma, Xie, and Geng, 2008) in terms of error measures and runtime and that using our order-independent skeleton recovery method turns the LCD algorithm into a stable method.

2.1 Basic Definitions and Concepts

In this chapter, we consider graphs containing both directed (→) and undirected (−) edges and largely use the terminology of (Lauritzen, 1996), where the reader can also find further details. Below we briefly list some of the central concepts used in this chapter.

If \( A \subseteq V \) is a subset of the vertex set in a graph \( G = (V, E) \), it induces a subgraph \( G_A = (A, E_A) \), where the edge set \( E_A = E \cap (A \times A) \) is obtained from \( G \) by keeping edges with both endpoints in \( A \). If there is an arrow from \( a \) pointing towards \( b \), \( a \) is said to be a parent of \( b \). The set of parents of \( b \) is denoted as \( pa(b) \). If there is an undirected edge between \( a \) and \( b \), \( a \) and \( b \) are said to be adjacent or neighbors. The set of neighbors of a vertex \( a \) is denoted as \( ne(a) \). The expressions \( pa(A) \) and \( ne(A) \) denote the collection of parents and neighbors of vertices in \( A \) that are not themselves elements of \( A \). The boundary \( bd(A) \) of a subset \( A \) of vertices is the set of vertices in \( V \setminus A \) that are parents or neighbors to vertices in \( A \). The closure of \( A \) is \( cl(A) = bd(A) \cup A \).

A path of length \( n \) from \( a \) to \( b \) is a sequence \( a = a_0, \ldots, a_n = b \) of distinct vertices such that \( (a_i, a_{i+1}) \in E \), for all \( i = 1, \ldots, n \). If there is a path from \( a \) to \( b \) we say that \( a \)
leads to $b$ and write $a \mapsto b$. The vertices $a$ such that $a \mapsto b$ and $b \not\mapsto a$ are the ancestors $an(b)$ of $b$. If $bd(a) \subseteq A$, for all $a \in A$ we say that $A$ is an ancestral set. The smallest ancestral set containing $A$ is denoted by $An(A)$. A chain of length $n$ from $a$ to $b$ is a sequence $a = a_0, \ldots, a_n = b$ of distinct vertices such that $(a_i, a_{i+1}) \in E$, or $(a_{i+1}, a_i) \in E$, or $\{a_i, a_{i+1}\} \in E$, for all $i = 1, \ldots, n$. It is called a cycle if $a_{n+1} \equiv a_0$, and $n \geq 3$. A chord of a cycle $C$ is an edge not in $C$ whose endpoints lie in $C$. A chordless cycle in $G$ is a cycle of length at least 4 in $G$ that has no chord (that is, the cycle is an induced subgraph). A cycle of length 3 is both chordal and chordless.

A partially directed cycle (or semi-directed cycle) in a graph $G$ is a sequence of $n$ distinct vertices $v_1, v_2, \ldots, v_n (n \geq 3)$, and $v_{n+1} \equiv v_1$, such that

(a) for all $i (1 \leq i \leq n)$ either $v_i - v_{i+1}$ or $v_i \rightarrow v_{i+1}$, and

(b) there exists a $j (1 \leq j \leq n)$ such that $v_j \rightarrow v_{j+1}$.

An LWF chain graph is a graph in which there are no partially directed cycles. The chain components $T$ of a chain graph are the connected components of the undirected graph obtained by removing all directed edges from the chain graph. A minimal complex (or simply a complex) in a chain graph is an induced subgraph of the form $a \rightarrow v_1 - \cdots - v_r \leftarrow b$. The skeleton (underlying graph) of an LWF CG $G$ is obtained from $G$ by changing all directed edges of $G$ into undirected edges.

We say that two LWF CGs $G$ and $H$ are Markov equivalent or that they are in the same Markov equivalence class if they induce the same conditional independence restrictions. Two chain graphs $G$ and $H$ are Markov equivalent if and only if they have the same skeletons and the same minimal complexes (Frydenberg, 1990). Every class of Markov equivalent CGs has a CG with the greatest number of undirected edges (or dually with the least number of directed edges). This graph is called the largest CG of the corresponding class of Markov equivalent CGs (Frydenberg, 1990).
Given an undirected graph $G$. Two vertices are said to be adjacent if they are connected by an edge. A subset $S \subseteq V$ that does not contain $a$ or $b$ is said to be an $(a, b)$-separator if all paths from $a$ to $b$ intersect $S$. A set $S$ of nodes that separates a given pair of nodes such that no proper subset of $S$ separates that pair is called a minimal separator. Note that removing an $(a, b)$-separator disconnects a graph into two connected components, one containing $a$, and another containing $b$. Conversely, if a set $S$ disconnects a graph into a connected component including $a$ and another connected component including $b$, then $S$ is an $(a, b)$-separator. Similarly, two disjoint vertex subsets $A$ and $B$ of $V$ are adjacent if there is at least one pair of adjacent vertices $u \in A$ and $v \in B$. Let $A$ and $B$ be two disjoint non-adjacent subsets of $V$. Similarly, we define an $(A, B)$-separator to be any subset of $V \setminus (A \cup B)$ whose removal separates $A$ and $B$ in distinct connected components. A minimal $(A, B)$-separator does not contain any other $(A, B)$-separator.

2.2 On the Properties of LWF Chain Graphs

Recall that an independence model $\perp$ is a ternary relation over subsets of a finite set $V$. The following properties have been defined for the conditional independencies of probability distributions. Note that every probability distribution $p$ satisfies the first four properties (Studený, 1989). Let $A, B, C, D$ be disjoint subsets of $V$ where $C$ may be the empty set.

- **S1 (Symmetry)** $A \perp B \mid C \implies B \perp A \mid C$;
- **S2 (Decomposition)** $A \perp BD \mid C \implies (A \perp B \mid C \text{ and } A \perp D \mid C)$;
- **S3 (Weak Union)** $A \perp BD \mid C \implies (A \perp B \mid DC \text{ and } A \perp D \mid BC)$;
- **S4 (Contraction)** $(A \perp B \mid DC \text{ and } A \perp D \mid C) \iff A \perp BD \mid C$;
- **S5 (Intersection)** $(A \perp B \mid DC \text{ and } A \perp D \mid BC) \implies A \perp BD \mid C$;
- **S6 (Composition)** $(A \perp B \mid C \text{ and } A \perp D \mid C) \iff A \perp BD \mid C$;
Let $G$ be a chain graph and $P$ be a probability measure defined on some product space $X = \times_{a \in V(G)} X_a$. Then $P$ satisfies the

**pairwise Markov property**, if for every pair $(v, u)$ of non-adjacent vertices with $u \in nd(v)$,

$$v \indep u \mid nd(v) \setminus \{v, u\}.$$  \hspace{1cm} (2.1)

**local Markov property**, relative to $G$, if for any vertex $v \in V(G)$,

$$v \indep nd(v) \setminus cl(v) \mid bd(v).$$ \hspace{1cm} (2.2)

**global Markov property**, relative to $G$, if for all $A, B, C \subseteq V$ such that $C$ separates $A$ and $B$ in $(G_{An(AU(Bu))})^m$, the moral graph of the smallest ancestral set containing $A \cup B \cup C$, we have $A \indep B \mid C$.

The factorization in the case of a chain graph involves two parts. Suppose $\{\tau : \tau \in D\}$ is the set of chain components of $G$. Then $P$ is said to factorize according to $G$ if it has density $f$ that satisfies:

(i) $f$ factorizes as in the directed acyclic case:

$$f(x) = \prod_{\tau \in D} f(x_\tau \mid x_{pa(\tau)}).$$

(ii) For each $\tau \in D$, $f$ factorizes in the moral graph of $G_{\tau \cup pa(\tau)}^m$:

$$f(x_\tau \mid x_{pa(\tau)}) = Z^{-1}(x_{pa(\tau)}) \prod_{h \in C} \psi_h(x)$$

where $C$ is the set of maximal cliques in $G_{\tau \cup pa(\tau)}^m$, $\psi_h(x)$ depends only on $x_h$ and

$$Z^{-1}(x_{pa(\tau)}) = \int_{X_\tau} \prod_{h \in C} \psi_h(x) \mu_\tau(dx_\tau).$$

If a probability measure $P$ factorizes according to $G$, then we say $P$ satisfies (CF). From arguments analogous to the directed and undirected cases, we have that in general

$$\text{(CF)} \implies \text{(CG)} \implies \text{(CL)} \implies \text{(CP)}.$$

If we assume (S5), then all Markov properties are equivalent.
Theorem 2.1. (Frydenberg, 1990) Assume that a probability measure $P$ defined on a chain graph $G$ is such that $(S5)$ holds for disjoint subsets of $V(G)$, then

$$(CF) \iff (CG) \iff (CL) \iff (CP).$$

2.3 Finding Minimal Separators in LWF Chain Graphs

In this section we propose and solve an optimization problem related to the separation in LWF chain graphs. The basic problem may be formulated as follows: given a pair of non-adjacent nodes, $x$ and $y$, in an LWF chain graph, $G$, find a minimal set of nodes that separates $x$ and $y$. We analyze several versions of this problem and offer polynomial time algorithms for each. These include the following problems:

Problem 1. (test for minimal separation) Given two non-adjacent nodes $X$ and $Y$ in an LWF chain graph $G$ and a set $Z$ that separates $X$ from $Y$, test if $Z$ is minimal i.e., no proper subset of $Z$ separates $X$ from $Y$.

Problem 2. (minimal separation) Given two non-adjacent nodes $X$ and $Y$ in an LWF chain graph $G$, find a minimal separating set between $X$ and $Y$, namely, find a set $Z$ such that $Z$, and no proper subset of $Z$, separates $X$ from $Y$.

Problem 3. (restricted separation) Given two non-adjacent nodes $X$ and $Y$ in an LWF chain graph $G$ and a set $S$ of nodes not containing $X$ and $Y$, find a subset $Z$ of $S$ that separates $X$ from $Y$.

Problem 4. (restricted minimal separation) Given two non-adjacent nodes $X$ and $Y$ in an LWF chain graph $G$ and a set $S$ of nodes not containing $X$ and $Y$, find a subset $Z$ of $S$ which is minimal and separates

Problem 5. (minimal separation of two disjoint non-adjacent sets) Given two disjoint non-adjacent sets $X$ and $Y$ in an LWF chain graph $G$, find a minimal separating set between $X$ and $Y$, namely, find a set $Z$ such that $Z$, and no proper subset of $Z$, separates $X$ from $Y$. 

13
Problem 6. (enumeration of all minimal separators) Given two non-adjacent nodes (or
disjoint subsets) $X$ and $Y$ in an LWF chain graph $G$, enumerate all minimal separating sets
between $X$ and $Y$.

We prove that it is possible to transform our problem into a separation problem, where
the undirected graph in which we have to look for the minimal set separating $X$ from $Y$
depends only on $X$ and $Y$. We propose and analyze an algorithm for each above mentioned
problem that, taking into account the previous results, solves it.

2.3.1 Main Theorem

In this subsection we prove that it is possible to transform our problem into a separation
problem, where the undirected graph in which we have to look for the minimal set separating $X$ from $Y$
depends only on $X$ and $Y$. Later, in the next subsections, we shall apply this
result to developing an efficient algorithm that solves our problems.

The next proposition shows that if we want to test a separation relationship between two
disjoint sets of nodes $X$ and $Y$ in an LWF chain graph, where the separating set is included
in the smallest ancestral set of $X \cup Y$, then we can test this relationship in a smaller chain
graph, whose set of nodes is formed only by the ancestors of $X$ and $Y$.

Proposition 2.2. Given an LWF chain graph $G = (V, E)$. Consider that $X$, $Y$, and $Z$ are three
disjoint subsets of $V$, and $Z \subseteq An(X \cup Y)$. Let $H = G_{An(X\cup Y)}$ be the subgraph of $G$ induced
by $An(X \cup Y)$. Then $\langle X, Y | Z \rangle_G \iff \langle X, Y | Z \rangle_H$.

Proof. ($\Rightarrow$) The necessary condition is obvious, because a separator in a graph is also a
separator in all of its subgraphs.

($\Leftarrow$) Let $\langle X, Y | Z \rangle_H$ and $Z \subseteq An(X \cup Y)$, then $An(X \cup Y \cup Z) = An(X \cup Y)$. Consider
that $\langle X, Y | Z \rangle_G$. This means that $X$ is not separated from $Y$ given $Z$ in $(G_{An(X\cup Y \cup Z)})^m = (G_{An(X\cup Y)})^m$. In other words, there is a chain $C$ between $X$ and $Y$ in $H^m = (G_{An(X\cup Y)})^m$ that
bypasses $Z$. Once again using $Z \subseteq An(X \cup Y)$, we obtain that $X$ and $Y$ are not separated by $Z$ in $H$, in contradiction to the assumption $\langle X, Y | Z \rangle_H$. Therefore, it has to be $\langle X, Y | Z \rangle_G$. □

The following proposition establishes the basic result necessary to solve our optimization problems.

**Proposition 2.3.** Given an LWF chain graph $G = (V, E)$. Consider that $X, Y$, and $Z$ are three disjoint subsets of $V$ such that $\langle X, Y | Z \rangle$ and $\langle X, Y \not\perp | Z' \rangle$, $\forall Z' \subsetneq Z$. Then $Z \subseteq An(X \cup Y)$.

**Proof.** Suppose that $Z \not\subseteq An(X \cup Y)$. Define $Z' = Z \cap An(X \cup Y)$. Then, by assumption we have $\langle X, Y | Z' \rangle$. Since $Z' \subseteq An(X \cup Y)$, it is obvious that $An(X \cup Y \cup Z') = An(X \cup Y)$. So, $X$ and $Y$ are not separated by $Z'$ in $(G_{An(X \cup Y)})^m$, hence there is a chain $C$ between $X$ and $Y$ in $(G_{An(X \cup Y)})^m$ that bypasses $Z'$ i.e., the chain $C$ is formed from nodes in $An(X \cup Y)$ that are outside of $Z$. Since $An(X \cup Y) \subseteq An(X \cup Y \cup Z)$, then $(G_{An(X \cup Y)})^m$ is a subgraph of $(G_{An(X \cup Y \cup Z)})^m$. Then, the previously found chain $C$ is also a chain in $(G_{An(X \cup Y \cup Z)})^m$ that bypasses $Z$, which means that $X$ and $Y$ are not separated by $Z$ in $(G_{An(X \cup Y \cup Z)})^m$, in contradiction to the assumption $\langle X, Y | Z \rangle$. Therefore, it has to be $Z \subseteq An(X \cup Y)$. □

The next proposition shows that, by combining the results in propositions 2.2 and 2.3, we can reduce our problems to a simpler one, which involves a smaller graph.

**Proposition 2.4.** Let $G = (V, E)$ be an LWF chain graph, and $X, Y \subseteq V$ are two disjoint subsets. Then the problem of finding a minimal separating set for $X$ and $Y$ in $G$ is equivalent to the problem of finding a minimal separating set for $X$ and $Y$ in the induced subgraph $G_{An(X \cup Y)}$.

**Proof.** The proof is very similar to the proof of Proposition 3 in (Acid and Campos, 1996). Let $H = G_{An(X \cup Y)}$, and let us to define sets $S_G = \{ Z \subseteq V | \langle X, Y | Z \rangle_G \}$ and $S_H = \{ Z \subseteq An(X \cup Y) | \langle X, Y | Z \rangle_H \}$. Then we have to prove that $\min_{Z \in S_G} |Z| = \min_{Z \in S_H} |Z|$, and therefore, by proposition 2.3, the sets of minimal separators are the same. From proposition 2.2, we deduce that $S_H \subseteq S_G$, and therefore $\min_{Z \in S_H} |Z| \geq \min_{Z \in S_G} |Z|$. 15
Let $T = \min(Z \in S_G)$. Then $\forall T' \subseteq T$ we have $T' \notin S_G$, and from proposition 2.3 we obtain $T \subseteq An(X \cup Y)$, and now using proposition 2.2 we get $T \in S_H$. So, we have $|T| = \min_{Z \in S_H} |Z| \geq \min_{Z \in S_G} |Z| = |T|$, hence $|T| = \min_{Z \in S_H} |Z|.

(\Rightarrow)$ Let $T = \min(Z \in S_H)$. If, $|T| = \min_{Z \in S_H} |Z| > \min_{Z \in S_G} |Z| = |Z_0|$, we have $\forall Z' \subsetneq Z_0, Z' \notin S_G$, and therefore, once again using proposition 2.3 and 2.2, we get $Z_0 \in S_H$, so that $|Z_0| \geq \min_{Z \in S_H} |Z| = |T|$, which is a contradiction. Thus, $|T| = \min_{Z \in S_G} |Z|$. □

**Theorem 2.5.** The problem of finding a minimal separating set for $X$ and $Y$ in an LWF chain graph $G$ is equivalent to the problem of finding a minimal separating set for $X$ and $Y$ in the undirected graph $(G_{An(X \cup Y)})^m$.

**Proof.** The proof is very similar to the proof of Theorem 1 in (Acid and Campos, 1996). Using the same notation from proposition 2.4, let $H^m$ be the moral graph of $H = G_{An(X \cup Y)}$, and $S_H^m = \{Z \subseteq An(X \cup Y) | \langle X, Y | Z \rangle_H^m \}$. Let $Z$ be any subset of $An(X \cup Y)$. Then taking into account the characteristics of ancestral sets, it is clear that $H_{An(X \cup Y) \cup Z} = H$. Then, we have $Z \in S_H \Leftrightarrow \langle X, Y | Z \rangle_H \Leftrightarrow \langle X, Y | Z \rangle_{(H_{An(X \cup Y) \cup Z})^m} \Leftrightarrow \langle X, Y | Z \rangle_{H^m} \Leftrightarrow Z \in S_H^m$.

![Figure 2.1: Test for minimal separation in LWF CGs](image)

Hence, $S_H = S_H^m$. Now, using proposition 2.4, we obtain $|T| = \min_{Z \in S_G} |Z| \Leftrightarrow |T| = \min_{Z \in S_H} |Z|$. □
2.3.2 Algorithms for Finding Minimal Separators

In undirected graphs we have efficient methods of testing whether a separation set is minimal, which are based on the following criterion.

**Theorem 2.6.** Given two nodes X and Y in an undirected graph, a separating set Z between X and Y is minimal if and only if for each node u in Z, there is a path from X to Y which passes through u and does not pass through any other nodes in Z.

**Proof.** See the proof of Theorem 5 in (Tian, Paz, and Pearl, 1998). □

This theorem leads to the Algorithm 1 for Problem 1. The idea is that if Z is minimal then all nodes in Z can be reached using Breadth First Search (BFS) that starts from both X and Y without passing any other nodes in Z.

---

**Algorithm 1:** Test for minimal separation (Problem 1)

**Input:** A set Z that separates two non-adjacent nodes X, Y in the LWF chain graph G.

**Output:** If Z is minimal then the algorithm returns TRUE otherwise, returns FALSE.

1. if Z contains a node that is not in An(X ∪ Y) then
2. return FALSE;
3. else
4. Construct G_{An(X∪Y)};
5. Construct (G_{An(X∪Y)})'';
6. Starting from X, run BFS. Whenever a node in Z is met, mark it if it is not already marked, and do not continue along that path. When BFS stops;
7. if not all nodes in Z are marked then
8. return FALSE;
9. else
10. Remove all markings. Starting from Y, run BFS. Whenever a node in Z is met, mark it if it is not already marked, and do not continue along that path. When BFS stops;
11. if not all nodes in Z are marked then
12. return FALSE;
13. else
14. return TRUE;
15. end
16. end
17. end
Analysis (Tian, Paz, and Pearl, 1998): Let \(|E^m_{An}|\) stands for the number of edges in 
\((G_{An(X∪Y)})^m\). Step 3-5 each requires \(O(|E^m_{An}|)\) time. Thus, the complexity of Algorithm 1 is 
\(O(|E^m_{An}|)\).

A variant of Algorithm 1 solves the Problem 2.

Algorithm 2: Minimal separation (Problem 2)

| Input: Two non-adjacent nodes X, Y in the LWF chain graph G. |
| Output: Set Z, that is a minimal separator for X, Y. |
1. Construct \(G'_{An(X∪Y)}\);  
2. Construct \((G'_{An(X∪Y)})^m\);  
3. Set \(Z'\) to be \(ne(X)\) (or \(ne(Y)\)) in \((G'_{An(X∪Y)})^m\);  
   /* \(Z'\) is a separator because, according to the local Markov  
   property of an undirected graph, a vertex is conditionally  
   independent of all other vertices in the graph, given its  
   neighbors (Lauritzen, 1996). */  
4. Starting from X, run BFS. Whenever a node in \(Z'\) is met, mark it if it is not already  
   marked, and do not continue along that path. When BFS stops, let \(Z''\) be the set of  
   nodes which are marked. Remove all markings;  
5. Starting from Y, run BFS. Whenever a node in \(Z''\) is met, mark it if it is not already  
   marked, and do not continue along that path. When BFS stops, let Z be the set of  
   nodes which are marked;  
6. return Z;  

Analysis: Step 2-5 each requires \(O(|E^m_{An}|)\) time. Thus, the overall complexity of Algo- 

rithm 2 is \(O(|E^m_{An}|)\).

Theorem 2.7. Given two nodes X and Y in an LWF chain graph G and a set S of nodes  
not containing X and Y, there exists some subset of S which separates X and Y if only if the  
set \(S' = S \cap An(X∪Y)\) separates X and Y.

Proof. \((⇒)\) Proof by contradiction. Let \(S' = S \cap An(X∪Y)\) and \(⟨X, Y \mid S'⟩\). Since \(S' \subseteq  
An(X∪Y)\), it is obvious that \(An(X∪Y∪S') = An(X∪Y)\). So, X and Y are not separated by \(S'\)  
in \((G_{An(X∪Y)})^m\), hence there is a chain \(C\) between X and Y in \((G_{An(X∪Y)})^m\) that bypasses \(S'\) i.e.,  
the chain \(C\) is formed from nodes in \(An(X∪Y)\) that are outside of \(S\). Since \(An(X∪Y) \subseteq  
An(X∪Y∪S''), \forall S'' \subseteq S\), then \((G_{An(X∪Y)})^m\) is a subgraph of \((G_{An(X∪Y∪S')})^m\). Then, the
previously found chain $C$ is also a chain in $(G_{\text{An}(X \cup Y \cup S'')})^m$ that bypasses $S''$, which means that $X$ and $Y$ are not separated by any $S'' \subseteq S$ in $(G_{\text{An}(X \cup Y \cup S'')})^m$, which is a contradiction.

$(\Leftarrow)$ It is obvious. \qed

Therefore, Problem 3 is solved by testing if $S' = S \cap \text{An}(X \cup Y)$ separates $X$ and $Y$.

**Algorithm 3:** Restricted separation (Problem 3)

**Input:** A set $S$ of nodes not containing $X$ and $Y$ in the LWF chain graph $G$.

**Output:** If there is a subset of $S$ that separates $X$ from $Y$ then the algorithm returns $Z \subseteq S$ that separates $X$ from $Y$ otherwise, returns FALSE.

1. Construct $G_{\text{An}(X \cup Y)}$;
2. Construct $(G_{\text{An}(X \cup Y)})^m$;
3. Set $S' = S \cap \text{An}(X \cup Y)$;
4. Remove $S'$ from $(G_{\text{An}(X \cup Y)})^m$;
5. Starting from $X$, run BFS;
6. if $Y$ is met then
5.1 return FALSE
7. else
5.2 return $Z = S'$
8. end
Analysis: This requires $O(|E_{Am}|)$ time.

According to Theorem 2.7, Problem 4 is solved using Algorithm 3 and then, if False not returned, Algorithm 2 with $Z' = S \cap An(X \cup Y)$. The time complexity of this algorithm is also $O(|E_{Am}|)$.

In order to solve Problem 5, i.e., to find the minimal set separating two disjoint non-adjacent subsets of nodes $X$ and $Y$ (instead of two single nodes) in an LWF chain graph $G$, first we build the undirected graph $(G_{Am(X\cup Y)})^m$. Next, starting out from this graph, we construct a new undirected graph $Aug[G : \alpha_X, \alpha_Y]$ by adding two artificial (dummy) nodes $\alpha_X, \alpha_Y$, and connect them to those nodes that are adjacent to some node in $X$ and $Y$, respectively. So, the separation of $X$ and $Y$ in $(G_{Am(X\cup Y)})^m$ is equivalent to the separation of $\alpha_X$ and $\alpha_Y$ in $Aug[G : \alpha_X, \alpha_Y]$. Moreover, the minimal separating set for $\alpha_X$ and $\alpha_Y$ in $Aug[G : \alpha_X, \alpha_Y]$ cannot contain nodes from $(X \cup Y)$. Therefore, in order to find the minimal separating set for $X$ and $Y$ in $G$, it is suffice to find the minimal separating set for $\alpha_X$ and $\alpha_Y$ in $Aug[G : \alpha_X, \alpha_Y]$. So, we have reduced this problem to one of separation for single nodes, which can be solved using the Algorithm 2.

Shen and Liang in (Shen and Liang, 1997) presents an efficient algorithm for enumerating all minimal $(X, Y)$-separators, separating given non-adjacent vertices $X$ and $Y$ in an undirected connected simple graph $G = (V, E)$. This algorithm requires $O(n^3R_{XY})$ time, where $|V| = n$ and $R_{XY}$ is the number of minimal $(X, Y)$-separators. The algorithm can be generalized for enumerating all minimal $(X, Y)$-separators that separate non-adjacent vertex sets $X, Y \subseteq V$, and it requires $O(n^2(n - n_X - n_Y)R_{XY})$ time. In this case, $|X| = n_X, |Y| = n_Y$, and $R_{XY}$ is the number of all minimal $(X, Y)$-separators. According to Theorem 2.5, using this algorithm for $(G_{Am(X\cup Y)})^m$ solves Problem 6.

Remark. Since DAGs (directed acyclic graphs) are subclass of chain graphs, one can use the same technique to enumerate all minimal separators in DAGs.
CONCLUSION AND SUMMARY

We have studied and solved the problem of finding minimal separating sets for pairs of variables in LWF chain graphs. We have also studied some extensions of the basic problem include finding a minimal separator from a restricted set of nodes, finding a minimal separator for two given disjoint sets, testing whether a given separator is minimal, and listing all minimal separators, given two non-adjacent nodes (or disjoint subsets) $X$ and $Y$ in an LWF chain graph $G$. Potential applications of this research include learning chain graphs from data and problems related to the selection of the variables to be instantiated when using chain graphs for inference tasks.

2.4 Efficient Learning of LWF Chain Graphs under the Faithfulness Assumption

LWF chain graphs were introduced by Lauritzen, Wermuth, and Frydenberg in the middle eighties to combine directed acyclic graphs (representing the structure of Bayesian networks) and undirected graphs (representing the structure of Markov networks). Every class of Markov equivalent chain graphs (that is, those chain graphs that induce the same conditional independence restrictions) has a unique natural representative, which is called the largest chain graph.

Figure 2.3: The procedure of learning the structure of an LWF chain graph from a faithful distribution.

As mentioned earlier, one important aspect of PGMs in general, and chain graphs es-
especially, is the possibility of learning the structure of models directly from sampled data. Four constraint-based learning algorithms, that use a statistical analysis to test the presence of a conditional independency, exist for learning LWF CGs: (1) the inductive causation like (IC-like) algorithm (Studený, 1997), (2) the decomposition-based algorithm called LCD (Learn Chain graphs via Decomposition) (Ma, Xie, and Geng, 2008), (3) the answer set programming (ASP) algorithm (Sonntag et al., 2015a), and (4) the inclusion optimal (CKES) algorithm (Peña, Sonntag, and Nielsen, 2014).

The learned graph of the LCD algorithm is only the pattern of an LWF CG, i.e. the graph that contains the same skeleton and complexes (a.k.a. U-structures). However, it may also contain semi-directed cycles. On the other hand, the IC-like algorithm finds the largest CG (that is, a CG with the greatest number of undirected edges) of the corresponding class of Markov equivalent CGs (that is, those chain graphs that induce the same conditional independence structure).

Similar to the inductive causation (IC) algorithm (Verma and Pearl, 1991), the IC-like algorithm (Studený, 1997) cannot be applied to large numbers of variables because for testing whether there is a set separating $X$ and $Y$ in the skeleton recovery, the IC-like algorithm might search all $2^{n-2}$ subsets of all $n$ random variables not including $X$ and $Y$. In order to overcome the scalability of the IC-like algorithm, we propose a constraint-based method for learning the structural of chain graphs based on the idea of the PC algorithm proposed by Peter Spirtes and Clark Glymour (Spirtes, Glymour, and Scheines, 2000), which is used for learning the structure of Bayesian networks (BNs). Our method modifies the IC-like algorithm to make it computationally feasible in the phase of skeleton recovery and to avoid the time consuming procedure of complex recovery.

We show that the proposed PC-like algorithm in this paper is order-dependent, in the sense that the output can depend on the order in which the variables are given. We propose several modifications of the PC-like algorithm that remove part or all of this order-dependence, but do not change the result when perfect conditional independence infor-
mation is used. When applied to data, the modified algorithms are partly or fully order-independent.

The R package `lcd` that implements the LCD algorithm uses an order-dependent skeleton recovery procedure for local skeleton recovery. We empirically show that using the order-independent version of the our proposed skeleton recovery algorithm for local skeleton recovery in the `lcd` package improves the precision of the LCD algorithm in high-dimensional settings.

2.4.1 PC-LIKE ALGORITHM

In this section, we discuss how the IC-like algorithm (Studený, 1997) can be modified to obtain a computationally feasible algorithm for LWF CGs recovery. A brief review of the IC-like algorithm is presented first, then we present a PC-like algorithm, which is a constraint-based algorithm that learns a CG from a probability distribution faithful to some CG.

The IC-like algorithm (Studený, 1997) is a constraint-based structural learning algorithm presented for LWF CGs and is based on three sequential phases. The first phase finds the adjacencies (skeleton recovery), the second phase orients the edges that must be oriented the same in every CG in the Markov equivalence class (complex recovery), and the third phase transforms this graph into a largest CG (LCG recovery).

The skeleton recovery of the IC-like algorithm works as follows: construct an undirected graph $H$ such that vertices $u$ and $v$ are connected with an undirected edge if and only if no set $S_{ub}$ can be found such that $u \perp \perp v|S_{ub}$. This procedure is very inefficient because this requires a number of independence tests that increases exponentially with the number of vertices. In other words, to determine whether there is a set separating $u$ and $v$, we might search all $2^{n-2}$ subsets of all $n$ random variables excluding $u$ and $v$. So, the complexity for investigating each possible edge in the skeleton is $O(2^n)$ and hence the complexity for constructing the skeleton is $O(n^2 2^n)$, where $n$ is the number of vertices in the LWF CG.
Since it is enough to find one $S$ making $u$ and $v$ independent to remove the undirected edge $u$ — $v$, one obvious short-cut is to do the tests in some order, and skip unnecessary tests. In the PC algorithm for BNs the revised edge-removal step is done as in the pseudocode at the top of the next column.

---

Since the PC algorithm only looks at adjacencies of $u$ and $v$ in the current stage of the algorithm, rather than all possible subsets, the PC algorithm performs fewer independence tests compared to the IC algorithm. The computational complexity of the PC algorithm for DAGs is difficult to evaluate exactly, but with the sparseness assumption the worst case is with high probability bounded by $O(n^q)$, where $n$ is the number of vertices and $q$ is the maximum number of the adjacent vertices of the true underlying DAG (Kalisch and Bühlmann, 2007). Our main intuition is that replacing the skeleton recovery phase in the IC-like algorithm with a PC-like approach will speed up this phase and make it computationally scalable when the true underlying LWF CG is sparse (see the skeleton recovery phase of Algorithm 4).

The looping procedure of the IC-like algorithm for complex recovery is computationally expensive. We use a polynomial time approach similar to the proposed algorithm by (Ma, Xie, and Geng, 2008) to reduce the computational cost of the complex recovery (see the complex recovery phase of Algorithm 4).
**Algorithm 4: PC-like algorithm for LWF CGs**

**Input:** a set $V$ of nodes and a probability distribution $p$ faithful to an unknown LWF CG $G$.

**Output:** The largest CG of the corresponding class of Markov equivalent CGs.

1. Let $H$ denote the complete undirected graph over $V$;

/* Skeleton Recovery */

2. for $i \leftarrow 0$ to $|V_H| - 2$ do

3. \hspace{1em} while possible do

4. \hspace{2em} Select any ordered pair of nodes $u$ and $v$ in $H$ such that $u \in ad_H(v)$ and $|ad_H(u) \setminus v| \geq i$;

5. \hspace{2em} /* $ad_H(x) := \{y \in V|x \rightarrow y, y \rightarrow x, \text{ or } x \rightarrow y\}$ */

6. \hspace{2em} if there exists $S \subseteq (ad_H(u) \setminus v)$ s.t. $|S| = i$ and $u \independent_v p|S$ (i.e., $u$ is independent of $v$ given $S$ in the probability distribution $p$) then

7. \hspace{3em} Set $S_{uv} = S_{vu} = S$;

8. \hspace{3em} Remove the edge $u - v$ from $H$;

9. \hspace{2em} end

10. end

/* Complex Recovery from (Ma, Xie, and Geng, 2008) */

11. Initialize $H^* = H$;

12. for each vertex pair $u, v$ s.t. $u$ and $v$ are not adjacent in $H$ do

13. \hspace{1em} for each $u - w$ in $H^*$ do

14. \hspace{2em} if $u \independent_v p|S_{uv} \cup \{w\}$ then

15. \hspace{3em} Orient $u - w$ as $u \rightarrow w$ in $H^*$;

16. \hspace{2em} end

17. end

18. end

/* To get the pattern of $H^*$ in line 19, at each step, we consider a pair of candidate complex arrows $u_1 \rightarrow w_1$ and $u_2 \rightarrow w_2$ with $u_1 \neq u_2$, then we check whether there is an undirected path from $w_1$ to $w_2$ such that none of its intermediate vertices is adjacent to either $u_1$ or $u_2$. If there exists such a path, then $u_1 \rightarrow w_1$ and $u_2 \rightarrow w_2$ are labeled (as complex arrows). We repeat this procedure until all possible candidate pairs are examined. The pattern is then obtained by removing directions of all unlabeled arrows in $H^*$ (Ma, Xie, and Geng, 2008). */

19. Take the pattern of $H^*$;
Finally, the IC-like algorithm uses three basic rules, namely the *transitivity rule*, the *necessity rule*, and the *double-cycle rule*, for changing the obtained pattern in the previous phase into the corresponding largest CG (see Studený, 1997) for details).

Both IC-like and LCD algorithms recover the structure of the model correctly if the probability distribution of the data is *faithful* to some LWF CGs i.e., all conditional independencies among variables can be represented by an LWF CG. The entire process is formally described in Algorithm 4. The correctness of Algorithm 4 is proved in Appendix B.

**Computational Complexity Analysis of Algorithm 4.** The complexity of the algorithm for a graph $G$ is bounded by the largest degree in $G$. Let $k$ be the maximal degree of any vertex and let $n$ be the number of vertices. Then in the worst case the number of conditional independence tests required by the algorithm is bounded by

$$2\binom{n}{2} \sum_{i=0}^{k} \binom{n-2}{i} \leq \frac{n^2(n-2)^k}{(k-1)!}$$

To derive the inequality, use induction on $k$ (Neapolitan, 2003, p. 552). So, algorithm 4 has a worst-case running time of $O(n^{k+2})$. This is a loose upper bound even in the worst case; it assumes that in the worst case for $n$ and $k$, no two variables are $c$-separated by a set of less than cardinality $k$, and for many values of $n$ and $k$ we have been unable to find graphs with that property. The worse case is rare, and the average number of conditional independence tests required for graphs of maximal degree $k$ is much smaller. In practice, our simulations show that it is possible to recover sparse graphs with a hundred variables in a few seconds.

**2.4.2 STABLE PC-LIKE ALGORITHM**

In this section, we show that the PC-like algorithm proposed in the previous section is order-dependent, in the sense that the output can depend on the order in which the variables are given.

In applications, we do not have perfect conditional independence information. Instead, we assume that we have an i.i.d. sample of size $n$ of variables $V = (X_1, \ldots, X_p)$. In the PC-
like algorithm all conditional independence queries are estimated by statistical conditional independence tests at some pre-specified significance level (p value) $\alpha$. For example, if the distribution of $V$ is multivariate Gaussian, one can test for zero partial correlation, see, e.g., Kalisch and Bühlmann (2007). Hence, we use the `gaussCItest()` function from the R package `pcalg` throughout this paper. Let $\text{order}(V)$ denote an ordering on the variables in $V$. We now consider the role of $\text{order}(V)$ in every step of the algorithm.

In the skeleton recovery phase of the PC-like algorithm, the order of variables affects the estimation of the skeleton and the separating sets. In particular, as noted for the special case of BNs in Colombo and Maathuis (2014), for each level of $i$, the order of variables determines the order in which pairs of adjacent vertices and subsets $S$ of their adjacency sets are considered (see lines 4 and 5 in Algorithm 4). The skeleton $H$ is updated after each edge removal. Hence, the adjacency sets typically change within one level of $i$, and this affects which other conditional independencies are checked, since the algorithm only conditions on subsets of the adjacency sets. When we have perfect conditional independence information, all orderings on the variables lead to the same output. In the sample version, however, we typically make mistakes in keeping or removing edges, because conditional independence relationships have to be estimated from data. In such cases, the resulting changes in the adjacency sets can lead to different skeletons, as illustrated in Example 1.

Moreover, different variable orderings can lead to different separating sets in the skeleton recovery phase. When we have perfect conditional independence information, this is not important, because any valid separating set leads to the correct $U$-structure decision in the complex recovery phase. In the sample version, however, different separating sets in the skeleton recovery phase may yield different decisions about $U$-structures in the complex recovery phase. This is illustrated in Example 2.

**Example 1. (Order-dependent skeleton of the PC-like algorithm.)** Suppose that the distribution of $V = \{a, b, c, d, e\}$ is faithful to the DAG in Figure 2.4(a). This DAG encodes the following conditional independencies with minimal separating sets: $a \perp d | \{b, c\}$ and
Suppose that we have an i.i.d. sample of \((a, b, c, d, e)\), and that the following conditional independencies with minimal separating sets are judged to hold at some significance level \(\alpha\): \(a \perp \perp d|\{b, c\}\), \(a \perp \perp e|\{b, c, d\}\), and \(c \perp \perp e|\{a, b, d\}\). Thus, the first two are correct, while the third is false.

We now apply the skeleton recovery phase of the PC-like algorithm with two different orderings: \(\text{order}_1(V) = (d, e, a, c, b)\) and \(\text{order}_2(V) = (d, c, e, a, b)\). The resulting skeletons are shown in Figures 2.4(b) and 2.4(c), respectively.

We see that the skeletons are different, and that both are incorrect as the edge \(c \rightarrow e\) is missing. The skeleton for \(\text{order}_2(V)\) contains an additional error, as there is an additional edge \(a \rightarrow e\). We now go through Algorithm 4 to see what happened. We start with a complete undirected graph on \(V\). When \(i = 0\), variables are tested for marginal independence, and the algorithm correctly does not remove any edge. Also, when \(i = 1\), the algorithm correctly does not remove any edge. When \(i = 2\), there is a pair of vertices that is thought to be conditionally independent given a subset of size two, and the algorithm correctly removes the edge between \(a\) and \(d\). When \(i = 3\), there are two pairs of vertices that are thought to be conditionally independent given a subset of size three. Table 2.1 shows the trace table of Algorithm 4 for \(i = 3\) and \(\text{order}_1(V) = (d, e, a, c, b)\). Table 2.2 shows the trace table of
Table 2.1: The trace table of Algorithm 4 for $i = 3$ and order 1 ($V = (d, e, a, c, b)$).

<table>
<thead>
<tr>
<th>Ordered Pair $(u, v)$</th>
<th>$\text{ad}_H(u)$</th>
<th>$S_{uv}$</th>
<th>Is $S_{uv} \subseteq \text{ad}_H(u) \setminus {v}$?</th>
<th>Is $u \rightarrow v$ removed?</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(e, a)$</td>
<td>${a, b, c, d}$</td>
<td>${b, c, d}$</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>$(e, c)$</td>
<td>${b, c, d}$</td>
<td>${a, b, d}$</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>$(c, e)$</td>
<td>${a, b, d, e}$</td>
<td>${a, b, d}$</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 2.2: The trace table of Algorithm 4 for $i = 3$ and order 2 ($V = (d, c, e, a, b)$).

<table>
<thead>
<tr>
<th>Ordered Pair $(u, v)$</th>
<th>$\text{ad}_H(u)$</th>
<th>$S_{uv}$</th>
<th>Is $S_{uv} \subseteq \text{ad}_H(u) \setminus {v}$?</th>
<th>Is $u \rightarrow v$ removed?</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(c, e)$</td>
<td>${a, b, d, e}$</td>
<td>${a, b, d}$</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>$(e, a)$</td>
<td>${a, b, d}$</td>
<td>${b, c, d}$</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>$(a, e)$</td>
<td>${b, c, e}$</td>
<td>${b, c, d}$</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Algorithm 4 for $i = 3$ and order 1 ($V = (d, e, a, c, b)$).

**Example 2. (Order-dependent separating sets and $U$-structures of the PC-like algorithm.)** Suppose that the distribution of $V = \{a, b, c, d, e\}$ is faithful to the DAG in Figure 2.5(a). This DAG encodes the following conditional independencies with minimal separating sets: $a \perp \perp d|b, a \perp \perp e|\{b, c\}, a \perp \perp e|\{c, d\}, b \perp \perp c, b \perp \perp e|d,$ and $c \perp \perp d.$

Suppose that we have an i.i.d. sample of $(a, b, c, d, e).$ Assume that all true conditional independencies are judged to hold except $c \perp \perp d.$ Suppose that $c \perp \perp d|b$ and $c \perp \perp d|e$ are thought to hold. Thus, the first is correct, while the second is false. We now apply the complex recovery phase of Algorithm 4 with two different orderings: order 1 ($V = (d, c, b, a, e)$) and order 2 ($V = (c, d, e, a, b)$). The resulting CGs are shown in Figures 2.5(b) and 2.5(c), respectively. Note that while the separating set for vertices $c$ and $d$ with order 1 is $S_{dc} = S_{cd} = \{b\},$ the separating set for them with order 2 is $S_{cd} = S_{dc} = \{e\}.$ This illustrates that order-dependent separating sets in the skeleton recovery phase of the sample version of Algorithm 4 can lead to order-dependent $U$-structures.

We now propose several modifications of the original PC-like algorithm (and hence...
Figure 2.5: (a) The DAG $G$, (b) the CG returned after the complex recovery phase of Algorithm 4 with order$_1(V)$, (c) the CG returned after the complex recovery phase of Algorithm 4 with order$_3(V)$.

also of the related algorithms) that remove the order-dependence in the various stages of the algorithm, analogously to what (Colombo and Maathuis, 2014) did for the original PC algorithm in the case of DAGs.

**ORDER-INDEPENDENT SKELETON RECOVERY**

We first consider estimation of the skeleton in the adjacency search of the PC-like algorithm. The pseudocode for our modification is given in Algorithm 5. The resulting algorithm is called *stable PC-like*.

The main difference between Algorithms 4 and 5 is given by the for-loop on lines 3-5 in the latter one, which computes and stores the adjacency sets $a_H(v_i)$ of all variables after each new size $i$ of the conditioning sets. These stored adjacency sets $a_H(v_i)$ are used whenever we search for conditioning sets of this given size $i$. Consequently, an edge deletion on line 10 no longer affects which conditional independencies are checked for other pairs of variables at this level of $i$. In other words, at each level of $i$, Algorithm 5 records which edges should be removed, but for the purpose of the adjacency sets it removes these edges only when it goes to the next value of $i$. Besides resolving the order-dependence in the estimation of the skeleton, our algorithm has the advantage that it is easily parallelizable at each level of $i$. The stable PC-like is correct, i.e. it returns an LWF CG to which the given probability distribution is faithful (Theorem 2.8), and it yields order-independent skeletons in the sample version (Theorem 2.9). We illustrate the algorithm in Example 3.
Algorithm 5: The order-independent (stable) PC-like algorithm for learning LWF CGs.

**Input:** A set $V$ of nodes and a probability distribution $p$ faithful to an unknown LWF CG $G$ and an ordering order($V$) on the variables.

**Output:** An LWF CG $G'$ s.t. $G$ and $G'$ are Markov equivalent and $G'$ has ....

1. Let $H$ denote the complete undirected graph over $V = \{v_1, \ldots, v_n\}$;

2. /* Skeleton Recovery */

   for $i \leftarrow 0$ to $|V_H| - 2$ do

   for $j \leftarrow 1$ to $|V_H|$ do

   Set $a_H(v_i) = a_H(v)$;

   end

   while possible do

   Select any ordered pair of nodes $u$ and $v$ in $H$ such that $u \in a_H(v)$ and $|a_H(u) \setminus v| \geq i$ using order($V$);

   if there exists $S \subseteq (a_H(u) \setminus v)$ s.t. $|S| = i$ and $u \independent v |S$ (i.e., $u$ is independent of $v$ given $S$ in the probability distribution $p$) then

   Set $S_{uv} = S_{vu} = S$;

   Remove the edge $u \longrightarrow v$ from $H$;

   end

   end

3. /* Complex Recovery and orientation rules */

4. Follow the same procedures in Algorithm 4 (lines: 11-19).

**Theorem 2.8.** Let the distribution of $V$ be faithful to an LWF CG $G$, and assume that we are given perfect conditional independence information about all pairs of variables $(u, v)$ in $V$ given subsets $S \subseteq V \setminus \{u, v\}$. Then the output of the stable PC-like algorithm is the pattern of $G$.

**Theorem 2.9.** The skeleton resulting from the sample version of the stable PC-like algorithm is order-independent.

**Example 3** (Order-independent skeletons). We go back to Example 1, and consider the sample version of Algorithm 4. The algorithm now outputs the skeleton shown in Figure 2.4(b) for both orderings order$_1(V)$ and order$_2(V)$. We again go through the algorithm step by step. We start with a complete undirected graph on $V$. No conditional independence found when $i = 0$. Also, when $i = 1$, the algorithm correctly does not remove any edge.
When \( i = 2 \), the algorithm first computes the new adjacency sets: \( a_H(v) = V \setminus \{v\}, \forall v \in V \). There is a pair of variables that is thought to be conditionally independent given a subset of size two, namely \((b, c)\). Since the sets \( a_H(v) \) are not updated after edge removals, it does not matter in which order we consider the ordered pair. Any ordering leads to the removal of edge between \( b \) and \( c \). When \( i = 3 \), the algorithm first computes the new adjacency sets: \( a_H(b) = a_H(c) = \{a, d, e\} \) and \( a_H(v) = V \setminus \{v\}, \) for \( v = a, d, e \). There are two pairs of variables that are thought to be conditionally independent given a subset of size three, namely \((a, e)\) and \((c, e)\). Since the sets \( a_H(v) \) are not updated after edge removals, it does not matter in which order we consider the ordered pair. Any ordering leads to the removal of both edges \( a \rightarrow e \) and \( c \rightarrow e \).

**Order-Independent Complex Recovery**

We propose two methods to resolve the order-dependence in the complex recovery phase, using the *conservative PC algorithm* (CPC) of Ramsey, Spirtes, and Zhang (2006) and the *majority rule PC-like algorithm* (MPC) of Colombo and Maathuis (2014).

The **Conservative PC-like algorithm** (CPC-like algorithm) works as follows. Let \( H \) be the undirected graph resulting from the skeleton recovery phase of Algorithm 4. For each vertex pair \( \{u, v\} \) s.t. \( u \) and \( v \) are not adjacent in \( H \), determine all subsets \( S \) of \( ad_H(u) \) that make \( u \) and \( v \) conditionally independent, i.e., that satisfy \( u \perp \!\!\!\perp v \mid S \). We refer to such sets as separating sets. The undirected edge \( u \rightarrow w \) is labelled as *unambiguous* if at least one such separating set is found and either for each \( S \) the set \( S \cup \{w\} \) c-separates \( u \) from \( v \) or for none of them \( S \cup \{w\} \) c-separates \( u \) from \( v \); otherwise it is labelled as *ambiguous*. If \( u \rightarrow w \) is unambiguous, it is oriented as \( u \rightarrow w \) if and only if for none of the separating sets \( S, S \cup \{w\} \) c-separates \( u \) from \( v \). Moreover, in the complex recovery phase of Algorithm 4, lines 11-18, the orientation rule is adapted so that only unambiguous undirected edges are oriented. The output of the CPC-like algorithm is a chain graph in which ambiguous undirected edges are marked. We refer to the combination of the stable PC-like and CPC-
like algorithms as the stable CPC-like algorithm.

In the case of DAGs, Colombo and Maathuis (2014) found that the CPC-algorithm can be very conservative, in the sense that very few unshielded triples ($\nu$-structures) are unambiguous in the sample version, where conditional independence relationships have to be estimated from data. They proposed a minor modification of the CPC approach, called Majority rule PC algorithm (MPC) to mitigate the (unnecessary) severity of CPC-like approach. We similarly propose the Majority rule PC-like algorithm (MPC-like) for LWF CGs. As in the CPC-like algorithm, we first determine all subsets $S$ of $\text{ad}_U(u)$ that make non adjacent vertices $u$ and $v$ conditionally independent, i.e., that satisfy $u \perp \!\!\!\perp v | S$. The undirected edge $u \leftarrow w$ is labelled as $(\alpha, \beta)$-unambiguous if at least one such separating set is found or no more than $\alpha\%$ or no less than $\beta\%$ of sets $S \cup \{w\}$ c-separate $u$ from $v$, for $0 \leq \alpha \leq \beta \leq 100$. Otherwise it is labelled as ambiguous. (As an example, consider $\alpha = 30$ and $\beta = 60$.) If an undirected edge $u \leftarrow w$ is unambiguous, it is oriented as $u \rightarrow w$ if and only if less than $\alpha\%$ of the sets $S \cup \{w\}$ c-separate $u$ from $v$. As in the CPC-like algorithm, the orientation rule in the complex recovery phase of the PC-like algorithm (Algorithm 4, lines 11-18) is adapted so that only unambiguous undirected edge $u \leftarrow w$ are oriented, and the output is a chain graph in which ambiguous undirected edge $u \leftarrow w$ are marked. Note that the CPC-like algorithm is the special case of the MPC-like algorithm with $\alpha = 0$ and $\beta = 100$. We refer to the combination of the stable PC-like and MPC-like algorithms as the stable MPC-like algorithm.

**Theorem 2.10.** Let the distribution of $V$ be faithful to an LWF CG $G$, and assume that we are given perfect conditional independence information about all pairs of variables $(u, v)$ in $V$ given subsets $S \subseteq V \setminus \{u, v\}$. Then the output of the (stable) CPC/MPC-like algorithm is the pattern of $G$.

**Theorem 2.11.** The decisions about $U$-structures in the sample version of the stable CPC / MPC-like algorithm are order-independent. In addition, the sample versions of stable CPC-like and stable MPC-like algorithms are fully order-independent.
2.4.3 Evaluation

In this section, we evaluate the performance of our algorithm in various setups using simulated/synthetic data sets. We first compare the performance of our algorithm with the LCD algorithm (Ma, Xie, and Geng, 2008) by running them on randomly generated LWF CGs (A brief description of the LCD algorithm is provided at the beginning of section 2.4.3). Empirical simulations show that our PC-like algorithm achieves competitive results with the LCD algorithm in terms of error measures and runtime.

We have also compared our method with the LCD algorithm on different discrete Bayesian networks such as ASIA, INSURANCE, ALARM, and HAILFINDER that have been widely used in evaluating the performance of structural learning algorithms. Algorithms has been implemented in R. All the results reported here are based on our R implementation. The R code and complete results are reported in our supplementary materials (Javidian, Valtorta, and Jamshidi, 2019d).

Performance Evaluation on Random LWF CGs

To investigate the performance of the proposed algorithms, we use the same approach as in (Ma, Xie, and Geng, 2008) for evaluating the performance of the LCD algorithm on LWF CGs. We run our algorithms, the LCD algorithm, and the stable LCD (SLCD) algorithm (which uses the same order-independent skeleton recovery procedure as Algorithm 5) on randomly generated LWF CGs and we compare the results and report summary error measures.

Data Generation Procedure First we explain the way in which the random LWF CGs and random samples are generated. Given a vertex set $V$, let $p = |V|$ and $N$ denote the average degree of edges (including undirected, pointing out, and pointing in) for each vertex. We generate a random LWF CG on $V$ as follows:

1. Order the $p$ vertices and initialize a $p \times p$ adjacency matrix $A$ with zeros;
2. For each element in the lower triangle part of $A$, set it to be a random number generated from a Bernoulli distribution with probability of occurrence $s = N/(p - 1)$;

3. Symmetrize $A$ according to its lower triangle;

4. Select an integer $k$ randomly from $\{1, \ldots, p\}$ as the number of chain components;

5. Split the interval $[1, p]$ into $k$ equal-length subintervals $I_1, \ldots, I_k$ so that the set of variables falling into each subinterval $I_m$ forms a chain component $C_m$;

6. Set $A_{ij} = 0$ for any $(i, j)$ pair such that $i \in I_l$, $j \in I_m$ with $l > m$.

This procedure yields an adjacency matrix $A$ for a chain graph with $(A_{ij} = A_{ji} = 1)$ representing an undirected edge between $V_i$ and $V_j$ and $(A_{ij} = 1, A_{ji} = 0)$ representing a directed edge from $V_i$ to $V_j$. Moreover, it is not difficult to see that $\mathbb{E}[\text{vertex degree}] = N$, where an adjacent vertex can be linked by either an undirected or a directed edge.

Given a randomly generated chain graph $G$ with ordered chain components $C_1, \ldots, C_k$, we generate a Gaussian distribution on it via the rnorm.cg function from the LCD R package.

**Experimental Results:** We evaluate the performance of the proposed algorithms in terms of the six measurements that are commonly used (Colombo and Maathuis, 2014; Kalisch and Bühlmann, 2007; Ma, Xie, and Geng, 2008; Tsamardinos et al., 2006) for constraint-based learning algorithms: (a) the true positive rate (TPR) (also known as sensitivity, recall, and hit rate), (b) the false positive rate (FPR) (also known as fall-out), (c) the true discovery rate (TDR) (also known as precision or positive predictive value), (d) accuracy (ACC) for the skeleton, (e) the structural Hamming distance (SHD) (this is the metric described in Tsamardinos et al. (2006) to compare the structure of the learned and the original graphs), and (f) run-time for the pattern recovery algorithms. In principle, large values of TPR, TDR, and ACC, and small values of FPR and SHD indicate good performance. In short, \[ TPR = \frac{\text{true positive (TP)}}{\text{the number of real positive cases in the data (Pos)}} \] is the ratio of the number of correctly
identified edges over total number of edges, \( FPR = \frac{\text{false positive (FP)}}{\text{the number of real negative cases in the data (Neg)}} \) is the ratio of the number of incorrectly identified edges over total number of gaps, \( TDR = \frac{\text{true positive (TP)}}{\text{the total number of edges in the recovered CG}} \) is the ratio of the number of correctly identified edges over total number of edges (both in estimated graph), \( ACC = \frac{\text{true positive (TP) + true negative (TN)}}{\text{Pos + Neg}} \) and \( SHD \) is the number of legitimate operations needed to change the current resulting graph to the true CG, where legitimate operations are: (a) add or delete an edge and (b) insert, delete or reverse an edge orientation. In principle, a large TPR, TDR, and ACC, a small FPR and SHD indicate good performance.

In our simulation, we change three parameters \( p \) (the number of vertices), \( n \) (sample size) and \( N \) (expected number of adjacent vertices) as follows:

- \( p \in \{50, 100\} \),
- \( n \in \{200, 2000\} \), and
- \( N \in \{2, 3\} \).

For each \((p, N)\) combination, we first generate 30 random LWF CGs. We then generate a random Gaussian distribution based on each graph and draw an identically independently distributed (i.i.d.) sample of size \( n \) from this distribution for each possible \( n \). For each sample, three different significance levels \((\alpha = 0.05, 0.005)\) are used to perform the hypothesis tests. The null hypothesis \( H_0 \) is “two variables \( u \) and \( v \) are conditionally independent given a set \( C \) of variables” and alternative \( H_1 \) is that \( H_0 \) may not hold. We then compare the results to access the influence of the significance testing level on the performance of our algorithms. Figures 2.6 and 2.7 show that: (a) as we expected (Ma, Xie, and Geng, 2008; Kalisch and Bühlmann, 2007), both algorithms work well on sparse graphs \((N = 2, 3)\), (b) for both algorithms, typically the TPR and TDR increase with sample size, (c) for both algorithms, typically the SHD decreases with sample size, (d) a large significance level \((\alpha = 0.05)\) typically yields large TPR and SHD, (e) in almost all cases, the performance of the decomposition-based algorithm based on all error measures is better than the per-
formance of the PC-like algorithm, and (f) generally, our empirical results suggests that in order to obtain a better performance, we can choose a small value (say \( \alpha = 0.005 \)) for the significance level of individual tests along with large sample (say \( n = 2000 \)). However, the optimal value for a desired overall error rate may depend on the sample size, significance level, and the sparsity of the underlying graph.

Since both the PC-like algorithm and the LCD algorithm assume faithfulness and the CKES algorithm (Peña, Sonntag, and Nielsen, 2014) does not assume the faithfulness requirement, the comparison between our proposed algorithms and the CKES algorithm may seem unfair (for a detailed discussion see (Peña, Sonntag, and Nielsen, 2014)). Also, we did not compare the proposed algorithms in this paper with the ASP algorithm due to the scalability issues discussed in (Sonntag et al., 2015a).

**Performance on Discrete Bayesian Networks**

Bayesian networks are special cases of LWF CGs. It is of interest to see whether the decomposition-based algorithms still work well when the data are actually generated from a Bayesian network. For this purpose, we perform simulation studies for four well-known Bayesian networks from Bayesian Network Repository: ASIA, INSURANCE, ALARM, and HAILFINDER. We purposefully selected these networks because they have different sizes (from small to large number of nodes, edges, and parameters). We briefly introduce these networks here:

- **ASIA** (Lauritzen and Spiegelhalter, 1988) with 8 nodes, 8 edges, and 18 parameters, it describes the diagnosis of a patient at a chest clinic who may have just come back from a trip to Asia and may be showing dyspnea. Standard learning algorithms are not able to recover the true structure of the network because of the presence of a functional node.

- **INSURANCE** (Binder et al., 1997) with 27 nodes, 52 edges, and 984 parameters, it evaluates car insurance risks.
Figure 2.6: Performance of the original and stable LCD and PC-like algorithms for randomly generated Gaussian chain graph models: average over 30 repetitions with 50 variables, expected degree $N = 3$, and significance levels $\alpha = 0.05, 0.005$. 

\begin{figure}
\centering
\includegraphics[width=\textwidth]{performance_graphs.png}
\caption{Performance of the original and stable LCD and PC-like algorithms for randomly generated Gaussian chain graph models: average over 30 repetitions with 50 variables, expected degree $N = 3$, and significance levels $\alpha = 0.05, 0.005$.}
\end{figure}
Figure 2.7: Performance of the original and stable LCD and PC-like algorithms for randomly generated Gaussian chain graph models: average over 30 repetitions with 100 variables, expected degree $N = 3$, and significance levels $\alpha = 0.05, 0.005$. 
• ALARM (Beinlich et al., 1989) with 37 nodes, 46 edges and 509 parameters, it was designed by medical experts to provide an alarm message system for intensive care unit patients based on the output a number of vital signs monitoring devices.

• HAILFINDER (Abramson et al., 1996) with 56 nodes, 66 edges, and 2656 parameters, it was designed to forecast severe summer hail in northeastern Colorado.

We compared the performance of our algorithms against the LCD algorithm for these Bayesian networks for three different significance levels ($\alpha = 0.05/0.01/0.005$). The results of all learning methods are summarized in Table 2.3. The results indicate that the performance of both algorithms in terms of FPR, ACC, and SHD are very similar. However, the LCD algorithm outperforms the PC-like algorithm in terms of the TPR.

**Discussion and Conclusion**

In this section, we presented a computationally feasible algorithm for learning the structure of LWF chain graphs. We compared the performance of our PC-like algorithm with that of the LCD algorithm proposed by (Ma, Xie, and Geng, 2008), in the Gaussian and discrete cases. Both PC-like algorithm and LCD algorithm are constraint-based algorithms that learn the structure of the underlying LWF chain graph in three steps: (a) determining the skeleton: the resulting undirected graph in this phase contains an undirected edge $u - v$ if there is no set $S \subseteq V \setminus \{u, v\}$ such that $u \perp \perp v|S$. The LCD algorithm achieves this by a divide-and-conquer approach; (b) determining the U-structures (minimal complexes). The LCD algorithm uses a localized search in this phase; (c) orienting some of the undirected edges into directed edges according to a set of rules applied iteratively to obtain the corresponding largest CG. The correctness of both algorithms lies upon the assumption that the probability distribution $p$ is faithful to some LWF CG. Empirical simulations in the Gaussian and discrete cases show that both algorithms yield good results when the underlying graph is sparse. The PC-like algorithm achieves competitive results with the
Table 2.3: Results for discrete samples from the ASIA, INSURANCE, ALARM, and HAILFINDER networks respectively. Each row corresponds to the significance level: $\alpha = 0.05/0.01/0.005$ respectively.

<table>
<thead>
<tr>
<th>TPR</th>
<th>FPR</th>
<th>ACC</th>
<th>SHD</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCD Algorithm</td>
<td>0.625</td>
<td>0.2</td>
<td>0.75</td>
</tr>
<tr>
<td></td>
<td>0.625</td>
<td>0.2</td>
<td>0.75</td>
</tr>
<tr>
<td></td>
<td>0.625</td>
<td>0.2</td>
<td>0.75</td>
</tr>
<tr>
<td>PC-Like Algorithm</td>
<td>0.625</td>
<td>0</td>
<td>0.893</td>
</tr>
<tr>
<td></td>
<td>0.625</td>
<td>0</td>
<td>0.893</td>
</tr>
<tr>
<td></td>
<td>0.625</td>
<td>0</td>
<td>0.893</td>
</tr>
<tr>
<td>LCD Algorithm</td>
<td>0.731</td>
<td>0.023</td>
<td>0.94</td>
</tr>
<tr>
<td></td>
<td>0.731</td>
<td>0.036</td>
<td>0.93</td>
</tr>
<tr>
<td></td>
<td>0.731</td>
<td>0.03</td>
<td>0.93</td>
</tr>
<tr>
<td>PC-Like Algorithm</td>
<td>0.62</td>
<td>0</td>
<td>0.94</td>
</tr>
<tr>
<td></td>
<td>0.65</td>
<td>0</td>
<td>0.95</td>
</tr>
<tr>
<td></td>
<td>0.62</td>
<td>0</td>
<td>0.94</td>
</tr>
<tr>
<td>LCD Algorithm</td>
<td>0.78</td>
<td>0.032</td>
<td>0.95</td>
</tr>
<tr>
<td></td>
<td>0.74</td>
<td>0.038</td>
<td>0.95</td>
</tr>
<tr>
<td></td>
<td>0.78</td>
<td>0.03</td>
<td>0.96</td>
</tr>
<tr>
<td>PC-Like Algorithm</td>
<td>0.43</td>
<td>0</td>
<td>0.96</td>
</tr>
<tr>
<td></td>
<td>0.46</td>
<td>0</td>
<td>0.96</td>
</tr>
<tr>
<td></td>
<td>0.41</td>
<td>0</td>
<td>0.96</td>
</tr>
<tr>
<td>LCD Algorithm</td>
<td>0.82</td>
<td>0.004</td>
<td>0.988</td>
</tr>
<tr>
<td></td>
<td>0.986</td>
<td>0.006</td>
<td>0.987</td>
</tr>
<tr>
<td></td>
<td>0.82</td>
<td>0.004</td>
<td>0.988</td>
</tr>
<tr>
<td>PC-Like Algorithm</td>
<td>0.515</td>
<td>0.0007</td>
<td>0.98</td>
</tr>
<tr>
<td></td>
<td>0.515</td>
<td>0.0007</td>
<td>0.98</td>
</tr>
<tr>
<td></td>
<td>0.515</td>
<td>0.0007</td>
<td>0.98</td>
</tr>
</tbody>
</table>

LCD learning algorithm in both Gaussian and discrete cases. In fact, the LCD method usually outperforms (in many cases slightly) the PC-like algorithm. We also provided stable (order-independent) versions of both algorithms and tested them.
Chapter 3

MVR Chain Graphs

Multivariate regression CGs (MVR CGs) were originally introduced by Cox and Wermuth (Cox and Wermuth, 1993, 1996) and widely studied in (Drton, 2009; Marchetti and Lupparelli, 2008, 2011; Wermuth and Cox, 2004; Sonntag and Peña, 2012, 2015a; Sonntag and Peña, 2015b).

Cox and Wermuth represented these graphs using directed edges and dashed edges, but we follow Richardson (Richardson, 2003) because bidirected edges allow the $m$-separation criterion (defined in section 3.2) to be viewed more directly as an extension of $d$-separation than is possible with dashed edges (Richardson, 2003). The use of bidirected edges is also more in keeping with the path-diagram notation introduced in (Wright, 1934). The most important difference about MVR CGs when compared to AMP CGs and LWF CGs is that MVR CG components contain bidirected instead of undirected edges. As a result, MVR CGs is a superclass of directed acyclic graphs (DAGs) and bidirected graphs (BGs), also known as covariance graphs, instead of DAGs and undirected graphs (UGs) as in the case of AMP and LWF CGs.

In this chapter, we address the problem of finding minimal separators in MVR chain graphs, because minimality is a desirable property to ensure efficiency and usability. Moreover, finding minimal separators is useful for learning and inference tasks (Acid and Campos, 1996; Tian, Paz, and Pearl, 1998).

As we mentioned in Chapter 1, the possibility of learning the structure of graphical models directly from sampled data is an important aspect of PGMs. So, in this chapter, we propose a decomposition-based algorithm for learning the structure of MVR chain graphs.
under the faithfulness assumption and we show that our proposed approach is comparable to the existing methods in terms of error measures and runtime.

The chapter is organized as follows. In section 3.1, we define the notation and terminology used throughout the chapter. In section 3.2, we take a closer look at MVR CGs in terms of separation criterion, Markov properties, factorization, Markov equivalence class, and essential chain graphs. In section 3.3, we propose and solve an optimization problem related to the separation in MVR chain graphs. In section 3.4 and 3.5, we consider the challenging task of recovering the structure of MVR CGs from sampled data. For this purpose, we consider the PC-like algorithm for structure learning of MVR CGs, a constraint-based method proposed by Sonntag and Peña in (Sonntag and Peña, 2012). We show that the PC-like algorithm is order-dependent, because the output can depend on the order in which the variables are given. This order-dependence is a minor issue in low-dimensional settings. However, it can be very pronounced in high-dimensional settings, where it can lead to highly variable results. We propose two modifications of the PC-like algorithm that remove part or all of this order-dependence. Also, we present a decomposition-based algorithm that finds the essential chain graph from data under the faithfulness assumption and we show that our proposed algorithm achieves competitive/better results with the existing methods in terms of error measures and runtime.

3.1 Basic Definitions and Concepts

In this subsection, we describe the notation and some basic concepts used throughout the chapter.

If $A \subseteq V$ is a subset of the vertex set in a graph $G = (V, E)$, it induces a subgraph $G_A = (A, E_A)$, where the edge set $E_A = E \cap (A \times A)$ is obtained from $G$ by keeping edges with both endpoints in $A$.

If there is an arrow from $a$ pointing towards $b$, $a$ is said to be a parent of $b$. The set of parents of $b$ is denoted as $pa(b)$. If there is an undirected edge between $a$ and $b$, $a$ and
are said to be adjacents. The set of adjacents of a vertex \( a \) is denoted as \( \text{adj}(a) \). The expressions \( \text{pa}(A) \) and \( \text{adj}(A) \) denote the collection of parents and adjacents of vertices in \( A \) that are not themselves elements of \( A \). The bound of a subset \( A \), \( \text{bound}(A) \), of vertices is the set of vertices in \( V \setminus A \) that are parents or adjacents to vertices in \( A \). The closure of \( A \) is \( \text{cl}(A) = \text{bound}(A) \cup A \).

A path of length \( n \) from \( a \) to \( b \) is a sequence \( a = a_0, \ldots, a_n = b \) of distinct vertices such that \( (a_i, a_{i+1}) \in E \), for all \( i = 1, \ldots, n \). A chain of length \( n \) from \( a \) to \( b \) is a sequence \( a = a_0, \ldots, a_n = b \) of distinct vertices such that \( (a_i, a_{i+1}) \in E \), or \( (a_{i+1}, a_i) \in E \), or \( \{a_i, a_{i+1}\} \in E \), for all \( i = 1, \ldots, n \). A vertex \( \alpha \) is said to be an ancestor of a vertex \( \beta \) if either there is a directed path \( \alpha \rightarrow \cdots \rightarrow \beta \) from \( \alpha \) to \( \beta \), or \( \alpha = \beta \). A vertex \( \alpha \) is said to be anterior to a vertex \( \beta \) if there is a path \( \mu \) from \( \alpha \) to \( \beta \) on which every edge is either of the form \( \gamma \leftarrow \delta \), or \( \gamma \rightarrow \delta \) with \( \delta \) between \( \gamma \) and \( \beta \), or \( \alpha = \beta \); that is, there are no edges \( \gamma \leftrightarrow \delta \) and there are no edges \( \gamma \leftarrow \delta \) pointing toward \( \alpha \). Such a path is said to be an anterior path from \( \alpha \) to \( \beta \). We apply these definitions disjunctively to sets: \( \text{an}(X) = \{\alpha | \alpha \text{ is an ancestor of } \beta \text{ for some } \beta \in X\} \), and \( \text{ant}(X) = \{\alpha | \alpha \text{ is an anterior of } \beta \text{ for some } \beta \in X\} \). If necessary we specify the graph by a subscript, as in \( \text{ant}_G(X) \). The usage of the terms “ancestor” and “anterior” differs from Lauritzen (Lauritzen, 1996), but follows Frydenberg (Frydenberg, 1990). A vertex \( \alpha \) is said to be antecedent to a vertex \( \beta \) if there is a path \( \mu \) from \( \alpha \) to \( \beta \) on which every edge is either of the form \( \gamma \leftrightarrow \delta \), or \( \gamma \rightarrow \delta \) with \( \delta \) between \( \gamma \) and \( \beta \), or \( \alpha = \beta \); that is, there are no edges of the form \( \gamma \leftarrow \delta \). Such a path is said to be an antecedent path from \( \alpha \) to \( \beta \). We apply this definition to sets: \( \text{antec}(X) = \{\alpha | \alpha \text{ is an antecedent of } \beta \text{ for some } \beta \in X\} \). If \( \text{antec}(a) \subseteq A \) for all \( a \in A \), we say that \( A \) is an antecedental set. The smallest antecedental set containing \( A \) is denoted by \( \text{Antec}(A) \).

**Definition 3.1.** A mixed graph is a graph containing three types of edges, undirected (−), directed (→) and bidirected (↔). An ancestral graph \( G \) is a mixed graph in which the following conditions hold for all vertices \( \alpha \) in \( G \):

(i) if \( \alpha \) and \( \beta \) are joined by an edge with an arrowhead at \( \alpha \), then \( \alpha \) is not anterior to \( \beta \).
(ii) there are no arrowheads present at a vertex which is an endpoint of an undirected edge.

**Definition 3.2.** A nonendpoint vertex $\zeta$ on a path is a *collider* on the path if the edges preceding and succeeding $\zeta$ on the path have an arrowhead at $\zeta$, that is, $\rightarrow \zeta \leftarrow$, or $\leftrightarrow \zeta \leftrightarrow$, or $\leftrightarrow \zeta \leftarrow$, or $\leftrightarrow \zeta \rightarrow$. A nonendpoint vertex $\zeta$ on a path which is not a collider is a noncollider on the path. A path between vertices $\alpha$ and $\beta$ in an ancestral graph $G$ is said to be $m$-connecting given a set $Z$ (possibly empty), with $\alpha, \beta \notin Z$, if:

(i) every noncollider on the path is not in $Z$, and

(ii) every collider on the path is in $\text{ant}_G(Z)$.

If there is no path $m$-connecting $\alpha$ and $\beta$ given $Z$, then $\alpha$ and $\beta$ are said to be $m$-separated given $Z$. Sets $X$ and $Y$ are $m$-separated given $Z$, if for every pair $\alpha, \beta$, with $\alpha \in X$ and $\beta \in Y$, $\alpha$ and $\beta$ are $m$-separated given $Z$ ($X, Y$, and $Z$ are disjoint sets; $X, Y$ are nonempty) and indicate this by $(X, Y|Z)_G$, where $G$ is omitted if clear from context. This criterion is referred to as a *global Markov property*. We denote the independence model resulting from applying the $m$-separation criterion to $G$, by $\mathcal{I}_m(G)$. This is an extension of Pearl’s $d$-separation criterion to mixed graphs in that in a DAG $D$, a path is $d$-connecting if and only if it is $m$-connecting.

Let $G_A$ denote the induced subgraph of $G$ on the vertex set $A$, formed by removing from $G$ all vertices that are not in $A$, and all edges that do not have both endpoints in $A$. Two vertices $x$ and $y$ in an MVR chain graph $G$ are said to be *collider connected* if there is a path from $x$ to $y$ in $G$ on which every non-endpoint vertex is a collider; such a path is called a *collider path*. (Note that a single edge trivially forms a collider path, so if $x$ and $y$ are adjacent in an MVR chain graph then they are collider connected.) The *augmented graph* derived from $G$, denoted $(G)^a$, is an undirected graph with the same vertex set as $G$ such that $c \rightarrow d$ in $(G)^a \iff c$ and $d$ are collider connected in $G$.

**Definition 3.3.** Disjoint sets $X, Y \neq \emptyset$, and $Z$ ($Z$ may be empty) are said to be $m^*$-separated if $X$ and $Y$ are separated by $Z$ in $(G^a_{\text{ant}(X \cup Y \cup Z)})^a$. Otherwise $X$ and $Y$ are said to be $m^*$-
Richardson and Spirtes in (Richardson and Spirtes, 2002, Theorem 3.18.) show that for an ancestral graph $G$, $\mathcal{I}_m(G) = \mathcal{I}_m^*(G)$. Note that in the case of ADMGs and MVR CGs, anterior sets in definitions 3.2, 3.3 can be replaced by ancestor sets, because in both cases anterior sets and ancestor sets are the same.

An ancestral graph $G$ is said to be maximal if for every pair of vertices $\alpha, \beta$ if $\alpha$ and $\beta$ are not adjacent in $G$ then there is a set $Z (\alpha, \beta \notin Z)$, such that $\langle\{\alpha\}, \{\beta\}|Z\rangle \in \mathcal{I}_m(G)$. Thus a graph is maximal if every missing edge corresponds to at least one independence in the corresponding independence model. A simple example of a nonmaximal ancestral graph is shown in Figure 3.1: $\gamma$ and $\delta$ are not adjacent, but are $m$-connected given every subset of $\{\alpha, \beta\}$, hence $\mathcal{I}_m(G) = \emptyset$. If $G$ is an undirected graph or a directed acyclic graph, then $G$ is a maximal ancestral graph (Richardson and Spirtes, 2002, Proposition 3.19). Richardson and Spirtes in (Richardson and Spirtes, 2002, Theorem 5.1) prove that if $G$ is an ancestral graph then there exists a unique maximal ancestral graph $G'$ formed by adding $\leftrightarrow$ edges to $G$ such that $\mathcal{I}_m(G) = \mathcal{I}_m(G')$. Therefore, from now on, without loss of generality we consider that given graphs in the above mentioned problems are maximal.

Acyclic directed mixed graphs (ADMGs), also known as semi-Markov(ian) (Pearl, 2009) models contain directed ($\rightarrow$) and bi-directed ($\leftrightarrow$) edges subject to the restriction that there are no directed cycles (Richardson, 2003; Evans and Richardson, 2014). An ADMG that has no partially directed cycle is called a *multivariate regression (MVR) chain*

![Figure 3.1: (Lauritzen and Richardson, 2002) A nonmaximal ancestral graph.](image-url)
ADMGs are a subclass of maximal ancestral graphs. In the next subsection, we prove that MVR chain graphs are a subclass of maximal ancestral graphs.

The absence of partially directed cycles in MVR CGs implies that the vertex set of a chain graph can be partitioned into so-called chain components such that edges within a chain component are bidirected whereas the edges between two chain components are directed and point in the same direction. So, any chain graph yields a directed acyclic graph $D$ of its chain components having $\mathcal{T}$ as a node set and an edge $T_1 \rightarrow T_2$ whenever there exists in the chain graph $G$ at least one edge $u \rightarrow v$ connecting a node $u$ in $T_1$ with a node $v$ in $T_2$. In this directed graph, we may define for each $T$ the set $\text{pa}_D(T)$ as the union of all the chain components that are parents of $T$ in the directed graph $D$. This concept is distinct from the usual notion of the parents $\text{pa}_G(A)$ of a set of nodes $A$ in the chain graph, that is, the set of all the nodes $w$ outside $A$ such that $w \rightarrow v$ with $v \in A$ (Marchetti and Lupparelli, 2011).

Given a chain graph $G$ with chain components $(T|T \in \mathcal{T})$, we can always define a strict total order $\prec$ of the chain components that is consistent with the partial order induced by the chain graph, such that if $T \prec T'$ then $T \notin \text{pa}_D(T')$ (we draw $T'$ to the right of $T$ as in the example of Figure 3.2).

![Figure 3.2: An MVR CG with chain components: $\mathcal{T} = \{T_1 = \{a, b\}, T_2 = \{c, d\}, T_3 = \{e, f\}, T_4 = \{g, h\}\}$.](Image)

For each $T$, the set of all components preceding $T$ is known and we may define the cumulative set $\text{pre}(T) = \cup_{T \prec T'} T'$ of nodes contained in the predecessors of component $T$, which we sometimes call the past of $T$. The set $\text{pre}(T)$ captures the notion of all the
potential explanatory variables of the response variables within $T$ (Marchetti and Lupparelli, 2011). In fact, MVR CGs can model the possible presence of residual associations among the responses using a bidirected graph, and this is consistent with an interpretation of bidirected edges in terms of latent variables (Roverato, 2017; Evans, 2016).

Given an undirected graph $G$. Two vertices are said to be adjacent if they are connected by an edge. A subset $S \subseteq V$ that does not contain $a$ or $b$ is said to be an $(a, b)$-separator if all paths from $a$ to $b$ intersect $S$. A set $S$ of nodes that separates a given pair of nodes such that no proper subset of $S$ separates that pair is called a minimal separator. Note that removing an $(a, b)$-separator disconnects a graph into two connected components, one containing $a$, and another containing $b$. Conversely, if a set $S$ disconnects a graph into a connected component including $a$ and another connected component including $b$, then $S$ is an $(a, b)$-separator. Two disjoint vertex subsets $A$ and $B$ of $V$ are adjacent if there is at least one pair of adjacent vertices $u \in A$ and $v \in B$. Let $A$ and $B$ be two disjoint non-adjacent subsets of $V$. Similarly, we define an $(A, B)$-separator to be any subset of $V \setminus (A \cup B)$ whose removal separates $A$ and $B$ in distinct connected components. A minimal $(A, B)$-separator does not contain any other $(A, B)$-separator.

3.2 On the Properties of MVR Chain Graphs

In the present subsection we deal with the Markov properties for an MVR chain graph $G = (V, E)$ thereby unifying the directed and bidirected cases.

Unlike in the other CG interpretations, the bidirected edge in MVR CGs has a strong intuitive meaning. It can be seen to represent one or more hidden common causes between the variables connected by it. In other words, in an MVR CG any bidirected edge $X \leftrightarrow Y$ can be replaced by $X \leftarrow H \rightarrow Y$ to obtain a Bayesian network representing the same independence model over the original variables, i.e. excluding the new variables $H$. These variables are called hidden, or latent, and have been marginalized away in the CG model (Sonntag, 2014).
Latent variables, which are often present in practice, cause several complications. First, causal inference based on structural learning (model selection) algorithms such as the PC algorithm (Spirtes, Glymour, and Scheines, 2000) may be incorrect. Second, if a distribution is faithful\(^1\) to a DAG, then the distribution obtained by marginalizing on some of the variables may not be faithful to any DAG on the observed variables, i.e., the space of DAGs is not closed under marginalization (Colombo et al., 2012).

**Example 4.** Consider that the DAG $G$ in Figure 3.3(a) is a perfect map of the distribution of $(X, Y, U, V, H)$, and suppose that $H$ is latent. There is no DAG on $\{X, Y, U, V\}$ that encodes exactly the same d-separation relations among $\{X, Y, U, V\}$ as $G$. Hence, there does not exist a perfect map of the marginal distribution of $(X, Y, U, V, H)$.

![Graphs](image)

**Figure 3.3:** (a) A directed graph including a vertex $H$ for an unobserved variable, (b) the independence structure encoded by the MVR CG

Mixed graphs provide a useful approach to address these problems without explicit modeling of latent variables (e.g., (Richardson and Spirtes, 2002; Pearl, 2009; Wermuth and Sadeghi, 2012)). The nodes of these graphs index the observed variables only. The edges, however, may be of two types, directed and bidirected. This added flexibility allows one to represent the more complicated dependence structures arising from a DAG with latent variables. A straightforward generalization of $d$-separation determines conditional independencies in mixed graph models (Drton and Maathuis, 2017). For instance, the MVR

---

\(^1\)A distribution $P$ is faithful to DAG $G$ if any independency in $P$ implies a corresponding $d$-separation property in $G$ (Spirtes, Glymour, and Scheines, 2000).
The causal interpretation of bidirected edges in MVR CGs along with the above discussion provide strong motivation for the importance of MVR CGs.
the equivalency of all proposed Markov properties for MVR chain graphs have not been proved, let alone carefully studied, in the literature, and this omission leads to confusion. The major contributions of this subsection may be summarized as follows:

- An alternative local Markov property for MVR chain graphs, which is equivalent to other Markov properties in the literature for compositional semi-graphoids.
- A comparison of different proposed Markov properties for MVR chain graphs in the literature and conditions under which they are equivalent.
- An alternative explicit factorization criterion for MVR chain graphs based on the proposed factorization criterion for acyclic directed mixed graphs in (Evans and Richardson, 2014).

3.2.1 **Markov Properties for MVR Chain Graphs**

In this section, first, we show, formally, that MVR chain graphs are a subclass of the maximal ancestral graphs of Richardson and Spirtes (Richardson and Spirtes, 2002) that include only observed and latent variables.

**Theorem 3.4.** If $G$ is an MVR chain graph, then $G$ is an ancestral graph.

**Proof.** Obviously, every MVR chain graph is a mixed graph without undirected edges. So, it is enough to show that condition (i) in Definition 3.1 is satisfied. For this purpose, consider that $\alpha$ and $\beta$ are joined by an edge with an arrowhead at $\alpha$ in MVR chain graph $G$. Two cases are possible. First, if $\alpha \leftrightarrow \beta$ is an edge in $G$, by definition of an MVR chain graph, both of them belong to the same chain component. Since all edges on a path between two nodes of a chain component are bidirected, then by definition $\alpha$ cannot be an anterior of $\beta$. Second, if $\alpha \leftarrow \beta$ is an edge in $G$, by definition of an MVR chain graph, $\alpha$ and $\beta$ belong to two different components ($\beta$ is in a chain component that is to the right side of the chain component that contains $\alpha$). We know that all directed edges in an MVR
chain graph are arrows pointing from right to left, so there is no path from \( \alpha \) to \( \beta \) in \( G \) i.e. \( \alpha \) cannot be an anterior of \( \beta \) in this case. We have shown that \( \alpha \) cannot be an anterior of \( \beta \) in both cases, and therefore condition (i) in Definition 3.1 is satisfied. In other words, every MVR chain graph is an ancestral graph.

\[ \square \]

The following result is often mentioned in the literature (Wermuth and Sadeghi, 2012; Peña, 2015; Sadeghi and Lauritzen, 2014; Sonntag, 2014), but we know of no published proof.

**Corollary 3.5.** Every MVR chain graph has the same independence model as a DAG under marginalization.

**Proof.** From Theorem 3.4, we know that every MVR chain graph is an ancestral graph. The result follows directly from (Richardson and Spirtes, 2002, Theorem 6.3).

\[ \square \]

**Corollary 3.6.** If \( G \) is an MVR chain graph, then \( G \) is a maximal ancestral graph.

**Proof.** To characterize maximal ancestral graphs, we need the following notion: A chain \(< r, q_1, \ldots, q_p, s >\) is a primitive inducing chain between \( r \) and \( s \) if and only if for every \( i \), \( 1 \leq i \leq p \):

- \( q_i \) is a collider on the chain; and
- \( q_i \in \text{an}(\{r\} \cup \{s\}) \).

Based on Corollary 4.4 in (Richardson and Spirtes, 2002), every nonmaximal ancestral graph contains a primitive inducing chain between a pair of nonadjacent vertices. So, it is enough to show that an MVR chain graph \( G \) does not contain a primitive inducing chain between any pair of nonadjacent vertices of \( G \). For this purpose, consider that \( r \) and \( s \) are a pair of nonadjacent vertices in MVR chain graph \( G \) such that chain \(< r, q_1, \ldots, q_p, s >\) is a primitive inducing chain between \( r \) and \( s \). So, for every \( i, 1 \leq i \leq p \): \( q_i \) is a collider on the
chain. Since, for every \( i, 1 \leq i \leq p: q_i \in an(r) \cup \{s\} \), there is a partially directed cycle in \( G \), which is a contradiction.

\[ \square \]

**Global and Pairwise Markov Properties**

The following properties have been defined for conditional independences of probability distributions. Let \( A, B, C \) and \( D \) be disjoint subsets of \( V_G \), where \( C \) may be the empty set.

1. **Symmetry**: \( A \perp \perp B \Rightarrow B \perp \perp A \);
2. **Decomposition**: \( A \perp \perp BD|C \Rightarrow (A \perp \perp B|C \text{ and } A \perp \perp D|C) \);
3. **Weak union**: \( A \perp \perp BD|C \Rightarrow (A \perp \perp B|DC \text{ and } A \perp \perp D|BC) \);
4. **Contraction**: \( (A \perp \perp B|DC \text{ and } A \perp \perp D|C) \Leftrightarrow A \perp \perp BD|C \);
5. **Intersection**: \( (A \perp \perp B|DC \text{ and } A \perp \perp D|BC) \Rightarrow A \perp \perp BD|C \);
6. **Composition**: \( (A \perp \perp B|C \text{ and } A \perp \perp D|C) \Rightarrow A \perp \perp BD|C \).

An independence model is a semi-graphoid if it satisfies the first four independence properties listed above. Note that every probability distribution \( p \) satisfies the semi-graphoid properties (Studený, 1989). If a semi-graphoid further satisfies the intersection property, we say it is a graphoid (Pearl and Paz, 1987; Studený, 2005, 1989). A compositional graphoid further satisfies the composition property (Sadeghi and Wermuth, 2016). If a semi-graphoid further satisfies the composition property, we say it is a compositional semi-graphoid.

For a node \( i \) in the connected component \( T \), its past, denoted by \( \text{pst}(i) \), consists of all nodes in components having a higher order than \( T \). To define pairwise Markov properties for MVR CGs, we use the following notation for parents, anteriors and the past of node pair \( i, j \) :

\[ \text{pa}_G(i, j) = \text{pa}_G(i) \cup \text{pa}_G(j) \setminus \{i, j\}, \text{ant}(i, j) = \text{ant}(i) \cup \text{ant}(j) \setminus \{i, j\}, \text{and pst}(i, j) = \text{pst}(i) \cup \text{pst}(j) \setminus \{i, j\} \].

The distribution \( \mathcal{P} \) of \( (X_n)_{n \in V} \) satisfies a pairwise Markov property \( (P_m) \), for \( m = 1, 2, 3, 4 \), with respect to MVR CG(\( G \)) if for every uncoupled pair of nodes \( i \) and \( j \) (i.e., there is no directed or bidirected edge between \( i \) and \( j \)):

\[ (P1): i \perp \perp j|\text{pst}(i, j), \]
\[ (P2): i \perp \perp j|\text{ant}(i, j), \]

53
(P3): \( i \perp \!\!\!\!\perp j | pa_G(i, j) \), and

(P4): \( i \perp \!\!\!\!\perp j | pa_G(i) \) if \( i < j \).

Notice that in (P4), \( pa_G(i) \) may be replaced by \( pa_G(j) \) whenever the two nodes are in the same connected component. Sadeghi and Wermuth in (Sadeghi and Wermuth, 2016) proved that all of above mentioned pairwise Markov properties are equivalent for compositional graphoids. Also, they show that each one of the above listed pairwise Markov properties is equivalent to the global Markov properties in Definitions 3.2, 3.3 (Sadeghi and Wermuth, 2016, Corollary 1). The necessity of intersection and composition properties follows from (Sadeghi and Lauritzen, 2014, Section 6.3).

**Block-recursive, Multivariate Regression (MR), and Ordered Local Markov Properties**

The following definition explains the meaning of the multivariate regression interpretation of a chain graph:

**Definition 3.7.** *(multivariate regression (MR) Markov property for MVR CGs (Marchetti and Lupparelli, 2011))*

Let \( G \) be a chain graph with chain components \( (T | T \in T) \), the set \( Nb_G(A) \) is the union of \( A \) itself and the set of nodes \( w \) that are neighbors of \( A \), that is, coupled by a bidirected edge to some node \( v \) in \( A \). Moreover, the set of non-descendants \( nd_D(T) \) of a chain component \( T \), is the union of all components \( T' \) such that there is no directed path from \( T \) to \( T' \) in the directed graph of chain components \( D \). A joint distribution \( P \) of the random vector \( X \) obeys the *multivariate regression (MR) Markov property* with respect to \( G \) if it satisfies the following independences. For all \( T \in T \) and for all \( A \subseteq T \):

(MR1) if \( A \) is connected: \( A \perp \!\!\!\!\perp [pre(T) \setminus pa_G(A)] | pa_G(A) \).

(MR2) if \( A \) is disconnected with connected components \( A_1, \ldots, A_r \): \( A_1 \perp \!\!\!\!\perp \ldots \perp A_r | pre(T) \).

\[ \text{A generalization of this property for regression graphs is the ordered regression graph Markov property in (Roverato, 2017).} \]
Remark 1. (Marchetti and Lupparelli, 2011, Remark 2) One immediate consequence of
Definition 3.7 is that if the probability density \( p(x) \) is strictly positive, then it factorizes ac-
cording to the directed acyclic graph of the chain components: 
\[
p(x) = \prod_{T \in \mathcal{T}} p(x_T | x_{\text{pa}_D(T)}).
\]

Drton discussed four different block-recursive Markov properties for chain graphs, of
which we discuss here those with the Markov property of type IV (Drton, 2009).

Definition 3.8. (Chain graph Markov property of type IV (Drton, 2009)) Let \( G \) be a chain
graph with chain components \((T | T \in \mathcal{T})\) and directed acyclic graph \( D \) of components. The
joint probability distribution of \( X \) obeys the block-recursive Markov property of type IV if
it satisfies the following independencies:

(IV0): \( T \perp \perp [\text{nd}_D(T) \setminus \text{pa}_D(T)] | \text{pa}_D(T) \), for all \( T \in \mathcal{T} \);
(IV1): \( A \perp \perp [\text{pa}_D(T) \setminus \text{pa}_G(A)] | \text{pa}_G(A) \), for all \( T \in \mathcal{T} \), and for all \( A \subseteq T \);
(IV2): \( A \perp \perp [T \setminus \text{Nb}_G(A)] | \text{pa}_D(T) \), for all \( T \in \mathcal{T} \), and for all connected subsets \( A \subseteq T \).

The following example shows that independence models, in general, resulting from
Definitions 3.7, 3.8 are different.

Example 5. Consider the MVR chain graph \( G \) in Figure 3.5. For the connected set \( A = \{1, 2\} \)
the condition (MR1) implies that \( 1, 2 \perp \perp 6, 7 | 5 \) while the condition (IV2) implies that
\( 1, 2 \perp \perp 6 | 5 \), which is not implied directly by (MR1) and (MR2). Also, the condition (MR2)
implies that \( 1 \perp \perp 3, 4 | 5, 6, 7 \) while the condition (IV2) implies that \( 1 \perp \perp 3, 4 | 5, 6 \), which is
not implied directly by (MR1) and (MR2).
Theorem 1 in (Marchetti and Lupparelli, 2011) states that for a given chain graph $G$, the multivariate regression Markov property is equivalent to the block-recursive Markov property of type IV. Also, Drton in (Drton, 2009, Section 7 Discussion) claims (without proof) that the block-recursive Markov property of type IV can be shown to be equivalent to the global Markov property proposed in (Richardson and Spirtes, 2002; Richardson, 2003).

Now, we introduce a local Markov property for ADMGs proposed by Richardson in (Richardson, 2003), which is an extension of the local well-numbering Markov property for DAGs introduced in (Lauritzen et al., 1990). For this purpose, we need to consider the following definitions and notations:

For a given acyclic directed mixed graph (ADMG) $G$, the induced bidirected graph $(G)\leftrightarrow$ is the graph formed by removing all directed edges from $G$. The district (aka c-component) for a vertex $x$ in $G$ is the connected component of $x$ in $(G)\leftrightarrow$, or equivalently

$$\text{dis}_G(x) = \{y \leftarrow \cdots \leftarrow x \text{ in } G, \text{ or } x = y\}.$$ 

As usual we apply the definition disjunctively to sets: $\text{dis}_A(B) = \bigcup_{x \in B} \text{dis}_A(x)$. A set $C$ is path-connected in $(G)\leftrightarrow$ if every pair of vertices in $C$ are connected via a path in $(G)\leftrightarrow$; equivalently, every vertex in $C$ has the same district in $G$.

In an ADMG, a set $A$ is said to be ancestrally closed if $x \rightarrow \cdots \rightarrow a$ in $G$ with $a \in A$ implies that $x \in A$. The set of ancestrally closed sets is defined as follows:

$$\mathcal{A}(G) = \{A\text{|} \text{an}_G(A) = A\}.$$ 

If $A$ is an ancestrally closed set in an ADMG $(G)$, and $x$ is a vertex in $A$ that has no children in $A$ then we define the Markov blanket of a vertex $x$ with respect to the induced subgraph on $A$ as

$$mb(x, A) = \text{pa}_G(\text{dis}_G(x)) \cup (\text{dis}_G(x) \setminus \{x\}),$$

where $\text{dis}_G(x)$ is the district of $x$ in the induced subgraph $G_A$. 

56
Let $G$ be an acyclic directed mixed graph. Specify a total ordering ($\prec$) on the vertices of $G$, such that $x \prec y \Rightarrow y \notin an(x)$; such an ordering is said to be consistent with $G$. Define $\text{pre}_{G, \prec}(x) = \{v | v < x \text{ or } v = x\}$.

**Definition 3.9** (Ordered local Markov property). Let $G$ be an acyclic directed mixed graph. An independence model $\mathcal{I}$ over the node set of $G$ satisfies the ordered local Markov property for $G$, with respect to the ordering $\prec$, if for any $x$, and ancestrally closed set $A$ such that $x \in A \subseteq \text{pre}_{G, \prec}(x)$,

$$\{x\} \perp \!\!\! \perp [A \setminus (\text{mb}(x, A) \cup \{x\})]|\text{mb}(x, A).$$

Since MVR chain graphs are a subclass of ADMGs, the ordered local Markov property in Definition 3.9 can be used as a local Markov property for MVR chain graphs.

Five of the Markov properties introduced in this and the previous subsection are equivalent for all probability distributions, as shown in the following theorem.

**Theorem 3.10.** Let $G$ be an MVR chain graph. For an independence model $\mathcal{I}$ over the node set of $G$, the following conditions are equivalent:

(i) $\mathcal{I}$ satisfies the global Markov property w.r.t. $G$ in Definition 3.2;

(ii) $\mathcal{I}$ satisfies the global Markov property w.r.t. $G$ in Definition 3.3;

(iii) $\mathcal{I}$ satisfies the block recursive Markov property w.r.t. $G$ in Definition 3.8;

(iv) $\mathcal{I}$ satisfies the MR Markov property w.r.t. $G$ in Definition 3.7.

(v) $\mathcal{I}$ satisfies the ordered local Markov property w.r.t. $G$ in Definition 3.9.

**Proof.** See Appendix A for the proof of this theorem. □

### 3.2.2 An Alternative Global Markov Property for MVR Chain Graphs

In this subsection we formulate an alternative global Markov property for MVR chain graphs. This property is different from the global Markov property resulting from the $m^*$-separation criterion proposed in (Richardson, 2003; Richardson and Spirtes, 2002). First,
show that this global Markov property implies the block-recursive Markov property of
type IV in (Drton, 2009; Marchetti and Lupparelli, 2008) and MR Markov property in
(Marchetti and Lupparelli, 2011). Finally, we show that they are equivalent.

**Definition 3.11.** (Alternative global Markov property for MVR chain graphs) For any triple
\((A, B, S)\) of disjoint subsets of \(V\) such that \(S\) separates \(A\) from \(B\) in \((G_{\text{Antec}(A \cup B \cup S)})^a\), in the
augmented graph of the smallest antecedental set containing \(A \cup B \cup S\), we have \(A \perp \perp B | S\).

**Proposition 3.12.** The alternative global Markov property in Definition 3.11 implies the
block-recursive Markov property in Definition 3.8.

**Proof.** The proof contains the three following steps:

1. Since \(\tau \cup nd_D(\tau)\) is an antecedental set, and \(pa_D(\tau)\) separates \(\tau\) from \(nd_D(\tau) \setminus pa_D(\tau)\)
in \((G_{\tau \cup nd_D(\tau)})^a\); this shows that the global Markov property in Definition 3.11 implies
(IV0) in Definition 3.8.

2. Assume that \(\sigma \subseteq \tau, \tau \in \mathcal{T}\). Consider that \(A\) is the smallest antecedental set containing
\(\sigma\) and \(pa_D(\tau)\). We know that for each vertex \(v \in pa_D(\tau) \setminus pa_G(\sigma)\), \(v \in \text{antec}(\sigma)\)
and \(pa_G(\sigma) \subseteq pa_D(\tau)\). Also, we know that there is no directed edge from \(pa_D(\tau) \setminus pa_G(\sigma)\)
to \(\sigma\) in \((G_A)^a\) has intersection with \(pa_G(\sigma)\), which means \(pa_G(\sigma)\) separates \(pa_D(\tau) \setminus pa_G(\sigma)\)
from \(\sigma\) in \((G_A)^a\); this shows that the global Markov property in Definition 3.11 implies (IV1) in Definition 3.8.

3. Assume that \(\sigma \subsetneq \tau, \tau \in \mathcal{T}\). Also, assume that \(\sigma\) is a connected subset of \(\tau\). Obviously,
\(\sigma\) and \(\tau \setminus Nb_G(\sigma)\) are two subsets of \(\tau\) such that there is no connection between
their elements. Consider that \(A\) is the smallest antecedental set containing \(\sigma\) and
\(\tau \setminus Nb_G(\sigma)\). Clearly, \(pa_D(\tau) \subseteq A\). Since \(\sigma\) and \(\tau \setminus Nb_G(\sigma)\) are disconnected in \(\tau\), any
connecting path between them (if it exists) must pass through \(pa_D(\tau)\) in \((G_A)^a\); this
shows that the global Markov property in Definition 3.11 implies (IV2) in Definition 3.8.

□

Proposition 3.13. The alternative global Markov property in Definition 3.11 implies the MR Markov property in Definition 3.7.

Proof. The proof is very similar to that of Proposition 3.12 and is omitted.

□

A special case of Definition 3.11 is the case in which $G$ is a DAG. In this case the augmented graph $(G_{\text{Antec}(A \cup B \cup S)})^a$ is the same as moral graph $(G_{\text{An}(A \cup B \cup S)})^m$. In fact, $A$ and $B$ are separated by $S$ if and only if $A$ and $B$ are $d$-separated by $S$. Thus we have the following corollary:

Corollary 3.14. In a DAG $\mathcal{D}$, $A$ is $d$-separated from $B$ given $S$ if and only if $S$ separates $A$ and $B$ in $(\mathcal{D}_{\text{Antec}(A \cup B \cup S)})^a$.

The following example shows that the proposed alternative global Markov property in Definition 3.11 is different from the pathwise $m$-separation criterion and the augmentation separation criterion in (Richardson, 2003; Richardson and Spirtes, 2002), in general.

Example 6. Consider the MVR chain graph $G$ in Figure 3.6.

![Figure 3.6: An MVR chain graph with chain components: $\mathcal{T} = \{a, b, c\}, \{d, e, f\}$.](image-url)
According to the pathwise m-separation criterion and the augmentation separation criterion in (Richardson, 2003; Richardson and Spirtes, 2002), $a$ and $f$ are marginally independent. However, according to Definition 3.11, $a$ and $f$ are not marginally independent but $a \perp \!\!\!\!\!\perp f|d$. Also, according to the pathwise $m$-separation criterion and the augmentation separation criterion in (Richardson, 2003; Richardson and Spirtes, 2002), $\{a, b\}$ and $f$ are marginally independent. However, according to Definition 3.11, we cannot obtain directly that $\{a, b\}$ and $f$ are marginally independent.

**Theorem 3.15.** Let $G$ be a MVR chain graph. An independence model $\mathcal{I}$ over the node set of $G$ satisfies the alternative global Markov property w.r.t. $G$ in Definition 3.11 if and only if it satisfies the global Markov property w.r.t. $G$ in Definition 3.3.

**Proof.** Assume that $S$ separates $A$ from $B$ in $(G_{\text{Antec}(A \cup B \cup S)})^a$, where $A, B, S$ are disjoint subsets of $V_G$. Since there is no undirected edge in a MVR CG, by definition of anterior and antecedent $\text{ant}(A \cup B \cup S)$ is a subgraph of $\text{Antec}(A \cup B \cup S)$. Therefore, $S$ separates $A$ from $B$ in $(G_{\text{ant}(A \cup B \cup S)})^a$. In other words, the independence model induced by the global Markov property w.r.t. $G$ in Definition 3.11 is a subset of the independence model induced by the global Markov property w.r.t. $G$ in Definition 3.3.

The result follows from Proposition 3.12 and Theorem 3.10. □

### 3.2.3 An Alternative Local Markov Property for MVR Chain Graphs

In this subsection we formulate an alternative local Markov property for MVR chain graphs. This property is different from and much more concise than the ordered Markov property proposed in (Richardson, 2003). The new local Markov property can be used to parameterize distributions efficiently when MVR chain graphs are learned from data, as done, for example, in (Javidian and Valtorta, 2019a, Lemma 9). While the new local Markov property is not equivalent to the five ones in Theorem 3.10 in general, we show that it is equivalent to the global and ordered local Markov properties of MVR chain graphs for compositional graphoids.
If there is a bidirected edge between vertices \( u \) and \( v \), \( u \) and \( v \) are said to be neighbors.

The boundary \( bd(u) \) of a vertex \( u \) is the set of vertices in \( V \setminus \{u\} \) that are parents or neighbors of vertex \( u \). The descendants of vertex \( u \) are \( de(u) = \{v | u \text{ is an ancestor of } v\} \). The non-descendants of vertex \( u \) are \( nd(u) = V \setminus (de(u) \cup \{u\}) \).

**Definition 3.16.** The local Markov property for an MVR chain graph \( G \) with vertex set \( V \) holds if, for every \( v \in V \): \( v \perp \!\!\!\!\perp [nd(v) \setminus bd(v)] | pa_G(v) \).

**Remark 2.** In DAGs, \( bd(v) = pa_G(v) \), and the local Markov property given above reduces to the directed local Markov property introduced by Lauritzen et al. in (Lauritzen et al., 1990). Also, in covariance graphs\(^3\) the local Markov property given above reduces to the dual local Markov property introduced by Kauermann in (Kauermann, 1996, Definition 2.1).

**Theorem 3.17.** Let \( G \) be an MVR chain graph. If an independence model \( \mathcal{I} \) over the node set of \( G \) is a compositional semi-graphoid, then \( \mathcal{I} \) satisfies the alternative local Markov property w.r.t. \( G \) in Definition 3.16 if and only if it satisfies the global Markov property w.r.t. \( G \) in Definition 3.3.

**Proof.** (Global \( \Rightarrow \) Local): Let \( X = \{v\} \), \( Y = nd(v) \setminus bd(v) \), and \( Z = pa_G(v) \). So, \( an(X \cup Y \cup S) = v \cup (nd(v) \setminus bd(v)) \cup pa_G(v) \) is an ancestor set, and \( pa_G(v) \) separates \( v \) from \( nd(v) \setminus bd(v) \) in \( (G \setminus (nd(v) \setminus bd(v)) \cup pa_G(v))^\nu \); this shows that the global Markov property in Definition 3.3 implies the local Markov property in Definition 3.16.

(Local \( \Rightarrow \) MR): We prove this by considering the following two cases:

**Case 1):** Let \( A \subseteq T \) is connected. Using the alternative local Markov property for each \( x \in A \) implies that: \( \{x\} \perp \!\!\!\!\perp [nd(x) \setminus bd(x)] | pa_G(x) \). Since \( (pre(T) \setminus pa_G(A)) \subseteq (nd(x) \setminus bd(x)) \), using the decomposition and weak union property give: \( \{x\} \perp \!\!\!\!\perp (pre(T) \setminus pa_G(A)) | pa_G(A) \), for all \( x \in A \). Using the composition property leads to (MR1): \( A \perp \!\!\!\!\perp (pre(T) \setminus pa_G(A)) | pa_G(A) \).

---

\(^3\)Equivalently, bidirected graphs, as explained in (Richardson, 2003, section 4.1).
Case 2): Let $A \subseteq T$ is disconnected with connected components $A_1, \ldots, A_r$. For $1 \leq i \neq j \leq r$ we have: $\{x\} \perp \{nd(x) \setminus bd(x)\}|pa_G(x)$, for all $x \in A_i$. Since $[(pre(T) \setminus pa_G(A)) \cup A_j] \subseteq (nd(x) \setminus bd(x))$, using the decomposition and weak union property give: $\{x\} \perp A_j|pre(T)$, for all $x \in A_i$. Using the composition property leads to (MR2): $A_i \perp A_j|pre(T)$, for all $1 \leq i \neq j \leq r$.

$(MR \Rightarrow Global)$: The result follows from Theorem 3.10. □

The necessity of composition property in Theorem 3.17 follows from the fact that local and global Markov properties for bidirected graphs, which are a subclass of MVR CGs, are equivalent only for compositional semi-graphoids (Kauermann, 1996; Banerjee and Richardson, 2003).

3.2.4 An Alternative Factorization for MVR Chain Graphs

According to the definition of MVR chain graphs, it is obvious that they are a subclass of acyclic directed mixed graphs (ADMGs). In this section, we derive an explicit factorization criterion for MVR chain graphs based on the proposed factorization criterion for acyclic directed mixed graphs in (Evans and Richardson, 2014). For this purpose, we need to consider the following definition and notations:

**Definition 3.18.** An ordered pair of sets $(H, T)$ form the head and tail of a term associated with an ADMG $G$ if and only if all of the following hold:

1. $H = \text{barren}(H)$, where $\text{barren}(H) = \{v \in H | \text{de}(v) \cap H = \{v\}\}$.
2. $H$ contained within a single district of $G_{\text{an}(H)}$.
3. $T = \text{tail}(H) = (\text{dis}_{\text{an}(H)}(H) \setminus H) \cup \text{pa}(\text{dis}_{\text{an}(H)}(H))$.

Evans and Richardson in (Evans and Richardson, 2014, Theorem 4.12) prove that a probability distribution $P$ obeys the global Markov property for an ADMG$(G)$ if and only if for every $A \in \mathcal{A}(G)$,

$$p(X_A) = \prod_{H \in [A]|H} p(X_H|\text{tail}(H)),$$  \hspace{1cm} (3.1)
where \([A]_G\) denotes a partition of \(A\) into sets \(\{H_1, \ldots, H_k\} \subseteq \mathcal{H}(G)\) (for a graph \(G\), the set of heads is denoted by \(\mathcal{H}(G)\)), defined with \(\text{tail}(H)\), as above. The following theorem provides an alternative factorization criterion for MVR chain graphs based on the proposed factorization criterion for acyclic directed mixed graphs in (Evans and Richardson, 2014).

**Theorem 3.19.** Let \(G\) be an MVR CG with chain components \((T|T \in \mathcal{T})\). If a probability distribution \(P\) obeys the global Markov property for \(G\) then 
\[
p(x) = \prod_{T \in \mathcal{T}} p(x_T | x_{\text{pa}_G(T)}).
\]

**Proof.** According to Theorem 4.12 in (Evans and Richardson, 2014), since \(G \in \mathcal{A}(G)\), it is enough to show that \(\mathcal{H}(G) = \{T|T \in \mathcal{T}\}\) and \(\text{tail}(T) = \text{pa}_G(T)\), where \(T \in \mathcal{T}\). In other words, it is enough to show that for every \(T\) in \(\mathcal{T}\), \((T, \text{pa}_G(T))\) satisfies the three conditions in Definition 3.18.

1. Let \(x, y \in T\) and \(T \in \mathcal{T}\). Then \(y\) is not a descendant of \(x\). Also, we know that \(x \in \text{de}(x)\), by definition. Therefore, \(T = \text{barren}(T)\).

2. Let \(T \in \mathcal{T}\), then from the definitions of an MVR chain graph and induced bidirected graph, it is obvious that \(T\) is a single connected component of the forest \((G_{\text{an}(T)})\rightarrow\). So, \(T\) contained within a single district of \((G_{\text{an}(T)})\rightarrow\).

3. \(T \subseteq \text{an}(T)\) by definition. So, \(\forall x \in T : \text{dis}_{an(T)}(x) = \{y \leftrightarrow \cdots \leftrightarrow x \text{ in } an(T), \text{ or } x = y\} = T\). Therefore, \(\text{dis}_{an(T)}(T) = T\) and \(\text{dis}_{an(T)}(T) \setminus T = \emptyset\). In other words, \(\text{tail}(T) = \text{pa}_G(T)\). □

**Example 7.** Consider the MVR chain graph \(G\) in Example 3.5. Since \([G]_G = \{(1, 2, 3, 4),\) \(\{5, 6, 7\} \} \) so, \(\text{tail}(\{1, 2, 3, 4\}) = \{5\}, \text{tail}(\{5, 6\}) = \{7\}, \text{and } \text{tail}(\{7\}) = \emptyset\). Therefore, based on Theorem 3.19 we have: 
\[
p = p_{1234|5} p_{56|7} p_7.
\]
However, the corresponding factorization of \(G\) based on the formula in (Drton, 2009; Marchetti and Lupparelli, 2011) is: 
\[
p = p_{1234|56} p_{56|7} p_7.
\]

The advantage of the new factorization is that it requires only graphical parents, rather than parent components in each factor, resulting in smaller variable sets for each factor, and therefore speeding up belief propagation. Moreover, the new factorization is the same as
the outer factorization of LWF and AMP CGs, as described in (Lauritzen, 1996; Lauritzen and Richardson, 2002; Cowell et al., 1999; Andersson, Madigan, and Perlman, 1996).

3.2.5 Intervention in MVR Chain Graphs

In the absence of a theory of intervention for chain graphs, a researcher would be unable to answer questions concerning the consequences of intervening in a system with the structure of a chain graph (Richardson, 1998). Fortunately, an intuitive account of the causal interpretation of MVR chain graphs is as follows. We interpret the edge $A \rightarrow B$ as $A$ being a cause of $B$. We interpret the edge $A \leftrightarrow B$ as $A$ and $B$ having an unobserved common cause $\lambda_{AB}$, i.e. a confounder.

Given the above causal interpretation of an MVR CG $G$, intervening on $X \subseteq V$ so that $X$ is no longer under the influence of its usual causes amounts to replacing the right-hand side of the equations for the random variables in $X$ with expressions that do not involve their usual causes and normalizing. Graphically, it amounts to modifying $G$ as follows. Delete from $G$ all the edges $A \rightarrow B$ and $A \leftrightarrow B$ with $B \in X$ (Peña, 2016a).

Conclusion and Summary

Based on the interpretation of the type of edges in a chain graph, there are different conditional independence structures among random variables in the corresponding probabilistic model. Other than pairwise Markov properties, we showed that for MVR chain graphs all Markov properties in the literature are equivalent for semi-graphoids. We proposed an alternative local Markov property for MVR chain graphs, and we proved that it is equivalent to other Markov properties for compositional semi-graphoids. Also, we obtained an alternative formula for factorization of an MVR chain graph.
3.3 Finding Minimal Separators in MVR Chain Graphs

In this section we propose and solve an optimization problem related to the separation in (maximal) ancestral graphs, which include MVR CGs as a special case (Theorem 3.4). (Tian, Paz, and Pearl, 1998) and (Acid and Campos, 1996) proposed efficient algorithms to find minimal $d$-separators in a directed acyclic graph (DAG). The motivation for our work is similar to that of (Tian, Paz, and Pearl, 1998): finding $d$-separators is necessary to apply the back-door criterion (Pearl, 2009), which is a way to estimate causal effects in causal Bayesian networks, and minimality is a desirable property to ensure efficiency and usability. We believe that the same method can be used in estimating causal effects with ancestral graphs, as suggested in (Malinsky and Spirtes, 2016). Of course, finding minimal separators is useful for other learning and inference tasks. For example, we used some of the results in this section for learning the structure of MVR CGs from sampled data (Javidian and Valtorta, 2019a).

The basic problem may be formulated as follows: given a pair of non-adjacent nodes, $x$ and $y$, in a maximal ancestral graph, $G$, find the set of nodes with minimum size that separates $x$ and $y$. We analyze several versions of this problem and offer polynomial time algorithms for each. These include the following problems:

**Problem 7.** (test for minimal separation) Given two non-adjacent nodes $X$ and $Y$ in a maximal ancestral graph $G$ and a set $Z$ that separates $X$ from $Y$, test if $Z$ is minimal i.e., no proper subset of $Z$ separates $X$ from $Y$.

**Problem 8.** (minimal separation) Given two non-adjacent nodes $X$ and $Y$ in a maximal ancestral graph $G$, find a minimal separating set between $X$ and $Y$, namely, find a set $Z$ such that $Z$, and no proper subset of $Z$, separates $X$ from $Y$.

**Problem 9.** (restricted separation) Given two non-adjacent nodes $X$ and $Y$ in a maximal ancestral graph $G$ and a set $S$ of nodes not containing $X$ and $Y$, find a subset $Z$ of $S$ that separates $X$ from $Y$. 

65
**Problem 10.** (restricted minimal separation) Given two non-adjacent nodes $X$ and $Y$ in a maximal ancestral graph $G$ and a set $S$ of nodes not containing $X$ and $Y$, find a subset $Z$ of $S$ which is minimal and separates

**Problem 11.** (minimal separation of two disjoint non-adjacent sets) Given two disjoint non-adjacent sets $X$ and $Y$ in a maximal ancestral graph $G$, find a minimal separating set between $X$ and $Y$, namely, find a set $Z$ such that $Z$, and no proper subset of $Z$, separates $X$ from $Y$.

**Problem 12.** (enumeration of all minimal separators) Given two non-adjacent nodes (or disjoint subsets) $X$ and $Y$ in a maximal ancestral graph $G$, enumerate all minimal separating sets between $X$ and $Y$.

We prove that it is possible to transform our problem into a separation problem, where the undirected graph in which we have to look for the minimal set separating $X$ from $Y$ depends only on $X$ and $Y$. We propose and analyze an algorithm for each above mentioned problem that, taking into account the previous results, solves it.

### 3.3.1 Main Theorem

In this subsection we prove that it is possible to transform our problem into a separation problem, where the undirected graph in which we have to look for the minimal set separating $X$ from $Y$ depends only on $X$ and $Y$. Later, in the next subsections, we shall apply this result to developing an efficient algorithm that solves our problems.

The next proposition shows that if we want to test a separation relationship between two disjoint sets of nodes $X$ and $Y$ in a maximal ancestral graph, where the separating set is included in the anterior set of $X \cup Y$, then we can test this relationship in a smaller maximal ancestral graph, whose set of nodes is formed only by the anteriors of $X$ and $Y$. \textit{Note that in all of the following propositions and theorem, the similar arguments can be applied...}
for acyclic directed mixed graphs and MVR chain graphs using ancestral set instead of anterior set.

**Proposition 3.20.** Let $G = (V, E)$ be a maximal ancestral graph, $X, Y, Z$ be three disjoint subsets of $V$, $Z \subseteq \text{ant}(X \cup Y)$, and $H = G_{\text{ant}(X \cup Y)}$ be the subgraph of $G$ induced by $\text{ant}(X \cup Y)$. Then $\langle X, Y | Z \rangle_G \Leftrightarrow \langle X, Y | Z \rangle_H$.

**Proof.** ($\Rightarrow$) The necessary condition is obvious, because a separator in a graph is also a separator in all of its subgraphs.

($\Leftarrow$) Let $\langle X, Y | Z \rangle_H$ and $Z \subseteq \text{ant}(X \cup Y)$, then $\text{ant}(X \cup Y \cup Z) = \text{ant}(X \cup Y)$. Consider that $\langle X, Y \not| Z \rangle_G$. This means that $X$ is not separated from $Y$ given $Z$ in $(G_{\text{ant}(X \cup Y \cup Z)})^a \equiv (G_{\text{ant}(X \cup Y)})^a$. In other words, there is a chain $C$ between $X$ and $Y$ in $H^a = (G_{\text{ant}(X \cup Y)})^a$ that bypasses $Z$. Once again using $Z \subseteq \text{ant}(X \cup Y)$, we obtain that $X$ and $Y$ are not separated by $Z$ in $H$, in contradiction to the assumption $\langle X, Y | Z \rangle_H$. Therefore, it has to be $\langle X, Y | Z \rangle_G$. □

The following proposition establishes the basic result necessary to solve our optimization problems.

**Proposition 3.21.** Let $G = (V, E)$ be a maximal ancestral graph, and $X, Y$, and $Z$ be three disjoint subsets of $V$ such that $\langle X, Y | Z \rangle$ and $\langle X, Y \not| Z' \rangle$, $\forall Z' \subseteq Z$. Then $Z \subseteq \text{ant}(X \cup Y)$.

**Proof.** Suppose that $Z \not\subseteq \text{ant}(X \cup Y)$. Define $Z' = Z \cap \text{ant}(X \cup Y)$. Then, by assumption we have $\langle X, Y \not| Z' \rangle$. Since $Z' \subseteq \text{ant}(X \cup Y)$, it is obvious that $\text{ant}(X \cup Y \cup Z') = \text{ant}(X \cup Y)$. So, $X$ and $Y$ are not separated by $Z'$ in $(G_{\text{ant}(X \cup Y)})^a$, hence there is a chain $C$ between $X$ and $Y$ in $(G_{\text{ant}(X \cup Y)})^a$ that bypasses $Z'$ i.e., the chain $C$ is formed from nodes in $\text{ant}(X \cup Y)$ that are outside of $Z$. Since $\text{ant}(X \cup Y) \subseteq \text{ant}(X \cup Y \cup Z)$, then $(G_{\text{ant}(X \cup Y)})^a$ is a subgraph of $(G_{\text{ant}(X \cup Y \cup Z)})^a$. Then, the previously found chain $C$ is also a chain in $(G_{\text{ant}(X \cup Y \cup Z)})^a$ that bypasses $Z$, which means that $X$ and $Y$ are not separated by $Z$ in $(G_{\text{ant}(X \cup Y \cup Z)})^a$, in contradiction to the assumption $\langle X, Y | Z \rangle$. Therefore, it has to be $Z \subseteq \text{ant}(X \cup Y)$. □
The next proposition shows that, by combining the results in propositions 2.2 and 2.3, we can reduce our problems to a simpler one, which involves a smaller graph.

**Proposition 3.22.** Let $G = (V, E)$ be a maximal ancestral graph, and $X, Y \subseteq V$ be two disjoint subsets. Then the problem of finding a minimal separating set for $X$ and $Y$ in $G$ is equivalent to the problem of finding a minimal separating set for $X$ and $Y$ in the induced subgraph $G_{\text{ant}(X \cup Y)}$.

**Proof.** The proof is very similar to the proof of Proposition 3 in (Acid and Campos, 1996). Let $H = G_{\text{ant}(X \cup Y)}$, and let us define sets $S_G = \{Z \subseteq V | \langle X, Y \mid Z \rangle_G \}$ and $S_H = \{Z \subseteq \text{ant}(X \cup Y) \langle X, Y \mid Z \rangle_H \}$. Then we have to prove that $|T| = \min_{Z \in S_G} |Z| \Leftrightarrow |T| = \min_{Z \in S_H} |Z|$, and therefore, by proposition 2.3, the sets of minimal separators are the same. From proposition 2.2, we deduce that $S_H \subseteq S_G$, and therefore $\min_{Z \in S_H} |Z| \geq \min_{Z \in S_G} |Z|$. 

$(\Rightarrow)$ If $|T| = \min_{Z \in S_G} |Z|$, then $\forall T' \subseteq T$ we have $T' \notin S_G$, and from proposition 2.3 we obtain $T \subseteq \text{ant}(X \cup Y)$, and now using proposition 2.2 we get $T \in S_H$. So, we have $|T| = \min_{Z \in S_H} |Z| \geq \min_{Z \in S_G} |Z| = |T|$, hence $|T| = \min_{Z \in S_H} |Z|$. 

$(\Leftarrow)$ If $|T| = \min_{Z \in S_H} |Z| > \min_{Z \in S_G} |Z| = |Z_0|$, we have $\forall Z' \subseteq Z_0, Z' \notin S_G$, and therefore, once again using proposition 2.3 and 2.2, we get $Z_0 \in S_H$, so that $|Z_0| \geq \min_{Z \in S_H} |Z| = |T|$, which is a contradiction. Thus, $|T| = \min_{Z \in S_G} |Z|$. $\Box$

**Theorem 3.23.** The problem of finding a minimal separating set for $X$ and $Y$ in a maximal ancestral graph $G$ is equivalent to the problem of finding a minimal separating set for $X$ and $Y$ in the undirected graph $(G_{\text{ant}(X \cup Y)})^a$.

**Proof.** The proof is very similar to the proof of Theorem 1 in (Acid and Campos, 1996). Using the same notation from proposition 2.4, let $H^a$ be the augmented graph of $H = G_{\text{ant}(X \cup Y)}$, and $S^a_H = \{Z \subseteq \text{ant}(X \cup Y) \langle X, Y \mid Z \rangle_{H^a} \}$. Let $Z$ be any subset of $\text{ant}(X \cup Y)$. Then taking into account the characteristics of ancestral sets, it is clear that $H_{\text{ant}(X \cup Y \cup Z)} = H$. 

68
Then, we have $Z \in S_H \iff \langle X, Y|Z \rangle_H \iff \langle X, Y|Z \rangle_{H_{ant(X \cup Y ; Z)}} \iff \langle X, Y|Z \rangle_{H^a} \iff Z \in S^a_H$.

Hence, $S_H = S^a_H$. Now, using proposition 2.4, we obtain $|T| = \min_{Z \in S_G} |Z| \iff |T| = \min_{Z \in S^a_H} |Z|$. □

### 3.3.2 Algorithms for Finding Minimal Separators

In undirected graphs we have efficient methods of testing whether a separation set is minimal, which are based on the following criterion (see Theorem 2.6): Given two nodes $X$ and $Y$ in an undirected graph, a separating set $Z$ between $X$ and $Y$ is minimal if and only if for each node $u$ in $Z$, there is a path from $X$ to $Y$ which passes through $u$ and does not pass through any other nodes in $Z$.

This criterion leads to the following algorithm for Problem 7. The idea is that if $Z$ is minimal then all nodes in $Z$ can be reached using Breadth First Search (BFS) that starts from both $X$ and $Y$ without passing any other nodes in $Z$.

*Note that in all of the following algorithms, the similar algorithm can be applied for acyclic directed mixed graphs and MVR chain graphs using ancestral set instead of anterior set.*

**Analysis (Tian, Paz, and Pearl, 1998):** Let $|E^a_{ant}|$ stands for the number of edges in $(G_{ant(X \cup Y)})^a$. Step 3-5 each requires $O(|E^a_{ant}|)$ time. Thus, the complexity of Algorithm 6 is $O(|E^a_{ant}|)$. 


Algorithm 6: Test for minimal separation (Problem 7)

Input: A set \( Z \) that separates two non-adjacent nodes \( X, Y \) in the maximal ancestral graph \( G \).

Output: If \( Z \) is minimal then the algorithm returns TRUE otherwise, returns FALSE.

1. if \( Z \) contains a node that is not in \( \text{ant}(X \cup Y) \) then
   2. return FALSE;
else
   4. Construct \( G_{\text{ant}(X \cup Y)} \);
   5. Construct \( (G_{\text{ant}(X \cup Y)})^a \);
   6. Starting from \( X \), run BFS. Whenever a node in \( Z \) is met, mark it if it is not already marked, and do not continue along that path. When BFS stops;
      7. if not all nodes in \( Z \) are marked then
         8. return FALSE;
      else
         10. Remove all markings. Starting from \( Y \), run BFS. Whenever a node in \( Z \) is met, mark it if it is not already marked, and do not continue along that path. When BFS stops;
            11. if not all nodes in \( Z \) are marked then
               12. return FALSE;
            else
               14. return TRUE;
         end
      end
   end
end

A variant of Algorithm 6 solves the Problem 8.

Analysis: Step 2-5 each requires \( O(|E_{\text{ant}}|) \) time. Thus, the overall complexity of Algorithm 7 is \( O(|E_{\text{ant}}|^a) \).

Consider that the distribution \( \mathcal{P} \) of \((X_n)_{n \in V}\) be a compositional graphoid. Then, for every valid ordering of the nodes of MVR chain graph \( G \) that \( i < j \) and \( i \) and \( j \) are non-adjacent vertices of \( V \), we have: \( \langle i, j \rangle|\text{pa}(i) \rangle \). Notice that in \( \langle i, j \rangle|\text{pa}(i) \rangle \), \( \text{pa}(i) \) may be replaced by \( \text{pa}(j) \) whenever the two nodes are in the same connected component (Sadeghi and Wermuth, 2016). Therefore, one can set \( Z' \) in step 3 of Algorithm 7 to be \( \text{pa}(X) \) (or \( \text{pa}(Y) \)), whenever the two nodes are in the same connected component).

Theorem 3.24. Given two nodes \( X \) and \( Y \) in a maximal ancestral graph \( G \) and a set \( S \) of nodes not containing \( X \) and \( Y \), there exists some subset of \( S \) which separates \( X \) and \( Y \) if
Algorithm 7: Minimal separation (Problem 8)

**Input:** Two non-adjacent nodes $X, Y$ in the maximal ancestral graph $G$.

**Output:** Set $Z$, that is a minimal separator for $X, Y$.

1. Construct $G_{ant}(X \cup Y)$;
2. Construct $(G_{ant}(X \cup Y))^a$;
3. Set $Z'$ to be $ne(X)$ (or $ne(Y)$) in $(G_{ant}(X \cup Y))^a$;
   /* $Z'$ is a separator because, according to the local Markov property of an undirected graph, a vertex is conditionally independent of all other vertices in the graph, given its neighbors (Lauritzen, 1996).
4. Starting from $X$, run BFS. Whenever a node in $Z'$ is met, mark it if it is not already marked, and do not continue along that path. When BFS stops, let $Z''$ be the set of nodes which are marked. Remove all markings;
5. Starting from $Y$, run BFS. Whenever a node in $Z''$ is met, mark it if it is not already marked, and do not continue along that path. When BFS stops, let $Z$ be the set of nodes which are marked;
6. return $Z$;

only if the set $S' = S \cap ant(X \cup Y)$ separates $X$ and $Y$.

**Proof.** ($\Rightarrow$) Proof by contradiction. Let $S' = S \cap ant(X \cup Y)$ and $\langle X, Y |\not \in S' \rangle$. Since $S' \subseteq ant(X \cup Y)$, it is obvious that $ant(X \cup Y \cup S') = ant(X \cup Y)$. So, $X$ and $Y$ are not separated by $S'$ in $(G_{ant}(X \cup Y))^a$, hence there is a chain $C$ between $X$ and $Y$ in $(G_{ant}(X \cup Y))^a$ that bypasses $S'$ i.e., the chain $C$ is formed from nodes in $ant(X \cup Y)$ that are outside of $S$. Since $ant(X \cup Y) \subseteq ant(X \cup Y \cup S'')$, $\forall S'' \subseteq S$, then $(G_{ant}(X \cup Y))^a$ is a subgraph of $(G_{ant}(X \cup Y \cup S''))^a$. Then, the previously found chain $C$ is also a chain in $(G_{ant}(X \cup Y \cup S''))^a$ that bypasses $S''$, which means that $X$ and $Y$ are not separated by any $S'' \subseteq S$ in $(G_{ant}(X \cup Y \cup S''))^a$, which is a contradiction.

($\Leftarrow$) It is obvious. □

Therefore, Problem 9 is solved by testing if $S' = S \cap ant(X \cup Y)$ separates $X$ and $Y$.

**Analysis:** This requires $O(|E_{ant}|)$ time.

According to Theorem 3.24, Problem 10 is solved using Algorithm 8 and then, if False not returned, Algorithm 7 with $Z' = S \cap ant(X \cup Y)$. The time complexity of this algorithm is also $O(|E_{ant}|)$.
Algorithm 8: Restricted separation (Problem 9)

Input: A set $S$ of nodes not containing $X$ and $Y$ in the maximal ancestral graph $G$.

Output: If there is a subset of $S$ that separates $X$ from $Y$ then the algorithm returns $Z \subseteq S$ that separates $X$ from $Y$ otherwise, returns FALSE.

1. Construct $G_{ant(X \cup Y)}$;
2. Construct $(G_{ant(X \cup Y)})^\mu$;
3. Set $S' = S \cap ant(X \cup Y)$;
4. Remove $S'$ from $(G_{ant(X \cup Y)})^\mu$;
5. Starting from $X$, run BFS;
6. if $Y$ is met then
   7.    return FALSE
8. else
   9.    return $Z = S'$
10. end

In order to solve Problem 11, i.e., to find the minimal set separating two disjoint non-adjacent subsets of nodes $X$ and $Y$ (instead of two single nodes) in a maximal ancestral graph $G$, first we build the undirected graph $(G_{ant(X \cup Y)})^\mu$. Next, starting out from this graph, we construct a new undirected graph $Aug[G : \alpha_X, \alpha_Y]$ by adding two artificial (dummy) nodes $\alpha_X, \alpha_Y$, and connect them to those nodes that are adjacent to some node in $X$ and $Y$, respectively. So, the separation of $X$ and $Y$ in $(G_{ant(X \cup Y)})^\mu$ is equivalent to the separation of $\alpha_X$ and $\alpha_Y$ in $Aug[G : \alpha_X, \alpha_Y]$. Moreover, the minimal separating set for $\alpha_X$ and $\alpha_Y$ in $Aug[G : \alpha_X, \alpha_Y]$ cannot contain nodes from $(X \cup Y)$. Therefore, in order to find the minimal separating set for $X$ and $Y$ in $G$, it is suffice to find the minimal separating set for $\alpha_X$ and
\(\alpha_Y\) in \(\text{Aug}[G : \alpha_X, \alpha_Y]\). So, we have reduced this problem to one of separation for single nodes, which can be solved using the Algorithm 7.

Shen and Liang in (Shen and Liang, 1997) presents an efficient algorithm for enumerating all minimal \((X, Y)\)-separators, separating given non-adjacent vertices \(X\) and \(Y\) in an undirected connected simple graph \(G = (V, E)\). This algorithm requires \(O(n^3 R_{XY})\) time, where \(|V| = n\) and \(R_{XY}\) is the number of minimal \((X, Y)\)-separators. The algorithm can be generalized for enumerating all minimal \((X, Y)\)-separators that separate non-adjacent vertex sets \(X, Y \subseteq V\), and it requires \(O(n^2(n - n_X - n_Y)R_{XY})\) time. In this case, \(|X| = n_X, |Y| = n_Y\), and \(R_{XY}\) is the number of all minimal \((X, Y)\)-separators. According to Theorem 3.23, using this algorithm for \((G_{\text{ant}(X \cup Y)})^{\alpha}\) solves Problem 12.

**Remark 3.** Since DAGs (directed acyclic graphs), MVR chain graphs, and acyclic directed mixed graph are subclass of ancestral graphs, one can use the same technique to enumerate all minimal separators in DAGs, MVR chain graphs, and acyclic directed mixed graphs.

**Conclusion and Summary**

We have studied and solved the problem of finding minimal separating sets for pairs of variables in maximal ancestral graphs. We have also studied some extensions of the basic problem include finding a minimal separator from a restricted set of nodes, finding a minimal separator for two given disjoint sets, testing whether a given separator is minimal, and listing all minimal separators, given two non-adjacent nodes (or disjoint subsets) \(X\) and \(Y\) in a maximal ancestral graph \(G\). Applications of this research include learning ancestral graphs from data (e.g., some of the results in this paper have been used for structural learning of MVR Chain Graphs (Javidian and Valtorta, 2019a)) and problems related to the selection of variables to be instantiated when using ancestral graphs for inference tasks, such as testing causal assumptions.
3.4 ORDER-INDEPENDENT STRUCTURE LEARNING OF MULTIVARIATE REGRESSION CHAIN GRAPHS

This section deals with multivariate regression chain graphs (MVR CGs), which were introduced by Cox and Wermuth in the nineties to represent linear causal models with correlated errors. Every class of Markov equivalent MVR chain graphs (that is, those MVR chain graphs that induce the same conditional independence restrictions) has a natural representative, which is called the essential chain graph.

**Definition 3.25.** A graph $G^*$ is said to be the essential MVR CG of a MVR CG $G$ if it has the same skeleton as $G$ and contains all and only the arrowheads common to every MVR CG in the Markov equivalence class of $G$.

An essential MVR CG does not need to be a MVR CG. Instead essential graphs can contain three types of edges, undirected, directed and bidirected. It can however be shown that no unique representation, that is a MVR CG, can exist for a Markov equivalence class of MVR CGs unless we assume some ordering of the nodes (Sonntag and Peña, 2015a). There is an algorithm for transforming an MVR chain graph into its essential graph (Sonntag, Peña, and Gómez-Olmedo, 2015).

Two constraint-based learning algorithms, that use a statistical analysis to test the presence of a conditional independency, exist for learning MVR CGs: (1) the PC-like algorithm (Sonntag and Peña, 2012), and (2) the answer set programming (ASP) algorithm (Peña, 2016b). The PC-like algorithm extends the original learning algorithm for Bayesian networks by Peter Spirtes and Clark Glymour (Spirtes, Glymour, and Scheines, 2000). It learns the structure of the underlying MVR chain graph in four steps: (a) determining the skeleton: the resulting undirected graph in this phase contains an undirected edge $u \prec v$ iff there is no set $S \subseteq V \setminus \{u, v\}$ such that $u \perp \!\!\!\perp v \mid S$; (b) determining the $\nu$-structures (unshielded colliders); (c) orienting some of the undirected/directed edges into directed/bidirected edges according to a set of rules applied iteratively; (d) transforming the resulting graph in the previous step into an MVR CG. The essential recovery algorithm
obtained after step (c) contains all directed and bidirected edges that are present in every MVR CG of the same Markov equivalence class.

In this section, we show that the PC-like algorithm is order-dependent, in the sense that the output can depend on the order in which the variables are given. We propose several modifications of the PC-like algorithm that remove part or all of this order-dependence, but do not change the result when perfect conditional independence information is used. When applied to data, the modified algorithms are partly or fully order-independent. Implementations in R and details of experimental results can be found at https://github.com/majavid/SUM2019.

3.4.1 Order-Dependent PC-like Algorithm

In this subsection, we show that the PC-like algorithm proposed by Sonntag and Peña in (Sonntag and Peña, 2012) is order-dependent, in the sense that the output can depend on the order in which the variables are given. The PC-like algorithm for learning MVR CGs under the faithfulness assumption is formally described in Algorithm 9.

In applications, we do not have perfect conditional independence information. Instead, we assume that we have an i.i.d. sample of size $n$ of variables $V = (X_1, \ldots, X_p)$. In the PC-like algorithm (Sonntag and Peña, 2012) all conditional independence queries are estimated by statistical conditional independence tests at some pre-specified significance level ($p$ value) $\alpha$. For example, if the distribution of $V$ is multivariate Gaussian, one can test for zero partial correlation, see, e.g., (Kalisch and Bühlmann, 2007). For this purpose, we use the `gaussCltest()` function from the R package `pcalg` throughout this section. Let
Algorithm 9: The order-dependent PC-like algorithm for learning MVR chain graphs
(Sonntag and Peña, 2012)

Input: A set $V$ of nodes and a probability distribution $p$ faithful to an unknown MVR CG $G$ and an ordering order$(V)$ on the variables.

Output: An MVR CG $G'$ s.t. $G$ and $G'$ are Markov equivalent and $G'$ has exactly the minimum set of bidirected edges for its equivalence class.

1. Let $H$ denote the complete undirected graph over $V$;

   /* Skeleton Recovery */

2. for $i \leftarrow 0$ to $|V_H| - 2$ do

   while possible do

   4. Select any ordered pair of nodes $u$ and $v$ in $H$ such that $u \in ad_H(v)$ and $|ad_H(u) \setminus v| \geq i$ using order$(V)$;

   5. if there exists $S \subseteq (ad_H(u) \setminus v)$ s.t. $|S| = i$ and $u \perp \perp v|S$ (i.e., $u$ is independent of $v$ given $S$ in the probability distribution $p$) then

   6. Set $S_{uv} = S_{vu} = S$;

   7. Remove the edge $u \rightarrow v$ from $H$;

   end

end

/* v-structure Recovery */

10. for each $m$-separator $S_{uv}$ do

   12. if $u \leftarrow w \rightarrow v$ appears in the skeleton and $w$ is not in $S_{uv}$ then

   13. Determine a $v$-structure $u \leftrightarrow w \leftarrow v$;

end

/* Essential Graph Recovery */

15. Apply rules 1-3 in Figure 3.9 while possible;

   /* After this line, the learned graph is the essential graph of MVR CG $G$. */

/* MVR Chain Graph Recovery */

17. Let $G'_u$ be the subgraph of $G'$ containing only the nodes and the undirected edges in $G'$;

18. Let $T$ be the junction tree of $G'_u$;

   /* If $G'_u$ is disconnected, the cliques belonging to different connected components can be linked with empty separators, as described in (Golumbic, 1980, Theorem 4.8). */

19. Order the cliques $C_1, \ldots, C_n$ of $G'_u$ s.t. $C_1$ is the root of $T$ and if $C_i$ is closer to the root than $C_j$ in $T$ then $C_i < C_j$;

20. Order the nodes such that if $A \in C_i$, $B \in C_j$, and $C_i < C_j$ then $A < B$;

21. Orient the undirected edges in $G'$ according to the ordering obtained in line 21.
order($V$) denote an ordering on the variables in $V$. We now consider the role of order($V$) in every step of the algorithm.

In the skeleton recovery phase of the PC-like algorithm (Sonntag and Peña, 2012), the order of variables affects the estimation of the skeleton and the separating sets. In particular, as noted for the special case of Bayesian networks in (Colombo and Maathuis, 2014), for each level of $i$, the order of variables determines the order in which pairs of adjacent vertices and subsets $S$ of their adjacency sets are considered (see lines 4 and 5 in Algorithm 9). The skeleton $H$ is updated after each edge removal. Hence, the adjacency sets typically change within one level of $i$, and this affects which other conditional independencies are checked, since the algorithm only conditions on subsets of the adjacency sets. When we have perfect conditional independence information, all orderings on the variables lead to the same output. In the sample version, however, we typically make mistakes in keeping or removing edges, because conditional independence relationships have to be estimated from data. In such cases, the resulting changes in the adjacency sets can lead to different skeletons, as illustrated in Example 8.

Moreover, different variable orderings can lead to different separating sets in the skeleton recovery phase. When we have perfect conditional independence information, this is not important, because any valid separating set leads to the correct $v$-structure decision in the orientation phase. In the sample version, however, different separating sets in the skeleton recovery phase of the algorithm may yield different decisions about $v$-structures in the orientation phase. This is illustrated in Example 9.

Finally, we consider the role of order($V$) on the orientation rules in the essential graph recovery phase of the sample version of the PC-like algorithm. Example 10 illustrates that different variable orderings can lead to different orientations, even if the skeleton and separating sets are order-independent.

**Example 8** (Order-dependent skeleton of the PC-like algorithm.). Suppose that the distribution of $V = \{a, b, c, d, e\}$ is faithful to the DAG in Figure 3.10(a). This DAG encodes the
following conditional independencies (using the notation defined in line 5 of Algorithm 9) with minimal separating sets: \( a \perp d | \{b, c\} \) and \( a \perp e | \{b, c\} \).

Suppose that we have an i.i.d. sample of \((a, b, c, d, e)\), and that the following conditional independencies with minimal separating sets are judged to hold at some significance level \( \alpha \): \( a \perp d | \{b, c\} \), \( a \perp e | \{b, c, d\} \), and \( c \perp e | \{a, b, d\} \). Thus, the first two are correct, while the third is false.

We now apply the skeleton recovery phase of the PC-like algorithm with two different orderings: \( \text{order}_1(V) = (d, e, a, c, b) \) and \( \text{order}_2(V) = (d, c, e, a, b) \). The resulting skeletons are shown in Figures 3.10(b) and 3.10(c), respectively.

Figure 3.10: (a) The DAG \( G \), (b) the skeleton returned by Algorithm 9 with \( \text{order}_1(V) \), (c) the skeleton returned by Algorithm 9 with \( \text{order}_2(V) \).

We see that the skeletons are different, and that both are incorrect as the edge \( c \rightarrow e \) is missing. The skeleton for \( \text{order}_2(V) \) contains an additional error, as there is an additional edge \( a \rightarrow e \). We now go through Algorithm 9 to see what happened. We start with a complete undirected graph on \( V \). When \( i = 0 \), variables are tested for marginal independence, and the algorithm correctly does not remove any edge. Also, when \( i = 1 \), the algorithm correctly does not remove any edge. When \( i = 2 \), there is a pair of vertices that is thought to be conditionally independent given a subset of size two, and the algorithm correctly removes the edge between \( a \) and \( d \). When \( i = 3 \), there are two pairs of vertices that are thought to
be conditionally independent given a subset of size three. Table 3.1 shows the trace table of Algorithm 9 for \( i = 3 \) and \( \text{order}_1(V) = (d, e, a, c, b) \).

Table 3.1: The trace table of Algorithm 9 for \( i = 3 \) and \( \text{order}_1(V) = (d, e, a, c, b) \).

<table>
<thead>
<tr>
<th>Ordered Pair ((u, v))</th>
<th>(ad_H(u))</th>
<th>(S_{uv})</th>
<th>Is (S_{uv} \subseteq ad_H(u) \setminus {v})?</th>
<th>Is (u \rightarrow v) removed?</th>
</tr>
</thead>
<tbody>
<tr>
<td>((e, a))</td>
<td>({a, b, c, d})</td>
<td>({b, c, d})</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>((e, c))</td>
<td>({b, c, d})</td>
<td>({a, b, d})</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>((c, e))</td>
<td>({a, b, d, e})</td>
<td>({a, b, d})</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 3.2 shows the trace table of Algorithm 9 for \( i = 3 \) and \( \text{order}_2(V) = (d, c, e, a, b) \).

Table 3.2: The trace table of Algorithm 9 for \( i = 3 \) and \( \text{order}_2(V) = (d, c, e, a, b) \).

<table>
<thead>
<tr>
<th>Ordered Pair ((u, v))</th>
<th>(ad_H(u))</th>
<th>(S_{uv})</th>
<th>Is (S_{uv} \subseteq ad_H(u) \setminus {v})?</th>
<th>Is (u \rightarrow v) removed?</th>
</tr>
</thead>
<tbody>
<tr>
<td>((c, e))</td>
<td>({a, b, d, e})</td>
<td>({a, b, d})</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>((e, a))</td>
<td>({a, b, d})</td>
<td>({b, c, d})</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>((a, e))</td>
<td>({b, c})</td>
<td>({b, c, d})</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

**Example 9** (Order-dependent separating sets and \(v\)-structures of the PC-like algorithm).

Suppose that the distribution of \( V = \{a, b, c, d, e\} \) is faithful to the DAG in Figure 3.11(a). This DAG encodes the following conditional independencies with minimal separating sets:

\[ a \perp \perp d | b, a \perp \perp e | \{b, c\}, a \perp \perp e | \{c, d\}, b \perp \perp c, b \perp \perp e | d, \text{and } c \perp \perp d. \]

Suppose that we have an i.i.d. sample of \((a, b, c, d, e)\). Assume that all true conditional independencies are judged to hold except \(c \perp \perp d\). Suppose that \(c \perp \perp d | b\) and \(c \perp \perp d | e\) are thought to hold. Thus, the first is correct, while the second is false. We now apply the \(v\)-structure recovery phase of the PC-like algorithm with two different orderings: \(\text{order}_1(V) = (d, c, b, a, e)\) and \(\text{order}_2(V) = (c, d, e, a, b)\). The resulting CGs are shown in Figures 3.11(b) and 3.11(c), respectively. Note that while the separating set for vertices \(c\) and \(d\) with \(\text{order}_1(V)\) is \(S_{dc} = S_{cd} = \{b\}\), the separating set for them with \(\text{order}_2(V)\) is \(S_{cd} = S_{dc} = \{e\}\).
Figure 3.11: (a) The DAG $G$, (b) the CG returned after the $v$-structure recovery phase of Algorithm 9 with order$_1(V)$, (c) the CG returned after the $v$-structure recovery phase of Algorithm 9 with order$_3(V)$.

This illustrates that order-dependent separating sets in the skeleton recovery phase of the sample version of the PC-algorithm can lead to order-dependent $v$-structures.

**Example 10** (Order-dependent orientation rules of the PC-like algorithm.). Consider the graph in Figure 3.12, and assume that this is the output of the sample version of the PC-like algorithm after $v$-structure recovery. Also, consider that $c \in S_{a,d}$ and $d \in S_{b,f}$. Thus, we have two $v$-structures, namely $a \to c \leftarrow e$ and $b \to d \leftarrow f$, and four unshielded triples, namely $(e, c, d), (c, d, f), (a, c, d)$, and $(b, d, c)$. Thus, we then apply the orientation rules in the essential recovery phase of the algorithm, starting with rule R1. If one of the two unshielded triples $(e, c, d)$ or $(a, c, d)$ is considered first, we obtain $c \to d$. On the other hand, if one of the unshielded triples $(b, d, c)$ or $(c, d, f)$ is considered first, then we obtain $c \leftarrow d$. Note that we have no issues with overwriting of edges here, since as soon as the edge $c \to d$ is oriented, all edges are oriented and no further orientation rules are applied. These examples illustrate that the essential graph recovery phase of the PC-like algorithm can be order-dependent regardless of the output of the previous steps.

Figure 3.12: Possible mixed graph after $v$-structure recovery phase of the sample version of the PC-like algorithm.
3.4.2 Order Independent Algorithms for Learning MVR CGs

We now propose several modifications of the original PC-like algorithm (and hence also of the related algorithms) that remove the order-dependence in the various stages of the algorithm, analogously to what Colombo and Maathuis (Colombo and Maathuis, 2014) did for the original PC algorithm in the case of DAGs. For this purpose, we discuss the skeleton, $\nu$-structures, and the orientation rules, respectively.

Order-Independent Skeleton Recovery

We first consider estimation of the skeleton in the adjacency search of the PC-like algorithm. The pseudocode for our modification is given in Algorithm 10. The resulting PC-like algorithm in Algorithm 10 is called stable PC-like.

The main difference between Algorithms 9 and 10 is given by the for-loop on lines 3-5 in the latter one, which computes and stores the adjacency sets $a_H(v_i)$ of all variables after each new size $i$ of the conditioning sets. These stored adjacency sets $a_H(v_i)$ are used whenever we search for conditioning sets of this given size $i$. Consequently, an edge deletion on line 10 no longer affects which conditional independencies are checked for other pairs of variables at this level of $i$.

In other words, at each level of $i$, Algorithm 10 records which edges should be removed, but for the purpose of the adjacency sets it removes these edges only when it goes to the next value of $i$. Besides resolving the order-dependence in the estimation of the skeleton, our algorithm has the advantage that it is easily parallelizable at each level of $i$. The stable PC-like algorithm is correct, i.e. it returns an MVR CG to which the given probability distribution is faithful (Theorem 3.26), and it yields order-independent skeletons in the sample version (Theorem 3.27). We illustrate the algorithm in Example 11.

**Theorem 3.26.** Let the distribution of $V$ be faithful to an MVR CG $G$, and assume that we are given perfect conditional independence information about all pairs of variables $(u, v)$
Algorithm 10: The order-independent (stable) PC-like algorithm for learning MVR chain graphs.

**Input:** A set \( V \) of nodes and a probability distribution \( p \) faithful to an unknown MVR CG \( G \) and an ordering \( \text{order}(V) \) on the variables.

**Output:** An MVR CG \( G' \) s.t. \( G \) and \( G' \) are Markov equivalent and \( G' \) has exactly the minimum set of bidirected edges for its equivalence class.

1. Let \( H \) denote the complete undirected graph over \( V = \{v_1, \ldots, v_n\} \);

2. for \( i \leftarrow 0 \) to \( |V_H| - 2 \) do
3.     for \( j \leftarrow 1 \) to \( |V_H| \) do
4.         Set \( a_H(v_i) = a_H(v_i); \)
5.     end
6.     while possible do
7.         Select any ordered pair of nodes \( u \) and \( v \) in \( H \) such that \( u \in a_H(v) \) and \( |a_H(u) \setminus v| \geq i \) using \( \text{order}(V) \);
8.         if there exists \( S \subseteq (a_H(u) \setminus v) \) s.t. \( |S| = i \) and \( u \perp \perp v|S \) (i.e., \( u \) is independent of \( v \) given \( S \) in the probability distribution \( p \)) then
9.             Set \( S_{uv} = S_{vu} = S; \)
10.            Remove the edge \( u \rightleftharpoons v \) from \( H; \)
11.        end
12.     end
13. /* \( v \)-structure Recovery and orientation rules */
14. Follow the same procedures in Algorithm 9 (lines: 11-21).

in \( V \) given subsets \( S \subseteq V \setminus \{u, v\} \). Then the output of the stable PC-like algorithm is an MVR CG that has exactly the minimum set of bidirected edges for its equivalence class.

**Proof.** The proof of Theorem 3.26 is completely analogous to the proof of Theorem 3 and 4 for the original PC-like algorithm in (Sonntag and Peña, 2012).

\[ \square \]

**Theorem 3.27.** The skeleton resulting from the sample version of the stable PC-like algorithm is order-independent.

**Proof.** We consider the removal or retention of an arbitrary edge \( u \rightleftharpoons v \) at some level \( i \). The ordering of the variables determines the order in which the edges (line 7 of Algorithm 10) and the subsets \( S \) of \( a_H(u) \) and \( a_H(v) \) (line 8 of Algorithm 10) are considered. By
construction, however, the order in which edges are considered does not affect the sets \( a_H(u) \) and \( a_H(v) \).

If there is at least one subset \( S \) of \( a_H(u) \) or \( a_H(v) \) such that \( u \perp \perp v|S \), then any ordering of the variables will find a separating set for \( u \) and \( v \). (Different orderings may lead to different separating sets as illustrated in Example 9, but all edges that have a separating set will eventually be removed, regardless of the ordering). Conversely, if there is no subset \( S' \) of \( a_H(u) \) or \( a_H(v) \) such that \( u \perp \perp v|S' \), then no ordering will find a separating set.

Hence, any ordering of the variables leads to the same edge deletions, and therefore to the same skeleton. \( \square \)

**Example 11** (Order-independent skeletons). We go back to Example 8, and consider the sample version of Algorithm 10. The algorithm now outputs the skeleton shown in Figure 3.10(b) for both orderings \( \text{order}_1(V) \) and \( \text{order}_2(V) \).

We again go through the algorithm step by step. We start with a complete undirected graph on \( V \). No conditional independence found when \( i = 0 \). Also, when \( i = 1 \), the algorithm correctly does not remove any edge. When \( i = 2 \), the algorithm first computes the new adjacency sets: \( a_H(v) = V \setminus \{v\}, \forall v \in V \). There is a pair of variables that is thought to be conditionally independent given a subset of size two, namely \((a, d)\). Since the sets \( a_H(v) \) are not updated after edge removals, it does not matter in which order we consider the ordered pair. Any ordering leads to the removal of edge between \( a \) and \( d \).

When \( i = 3 \), the algorithm first computes the new adjacency sets: \( a_H(a) = a_H(d) = \{b, c, e\} \) and \( a_H(v) = V \setminus \{v\}, \forall v = b, c, e \). There are two pairs of variables that are thought to be conditionally independent given a subset of size three, namely \((a, e)\) and \((c, e)\). Since the sets \( a_H(v) \) are not updated after edge removals, it does not matter in which order we consider the ordered pair. Any ordering leads to the removal of both edges \( a \rightarrow e \) and \( c \rightarrow e \).
We propose two methods to resolve the order-dependence in the determination of the $v$-structures, using the conservative PC algorithm (CPC) of Ramsey et al. (Ramsey, Spirtes, and Zhang, 2006) and the majority rule PC-like algorithm (MPC) of Colombo & Maathuis (Colombo and Maathuis, 2014).

The **Conservative PC-like algorithm (CPC-like algorithm)** works as follows. Let $H$ be the undirected graph resulting from the skeleton recovery phase of the PC-like algorithm (Algorithm 9). For all unshielded triples $(X_i, X_j, X_k)$ in $H$, determine all subsets $S$ of $ad_H(X_i)$ and of $ad_H(X_k)$ that make $X_i$ and $X_k$ conditionally independent, i.e., that satisfy $X_i \perp \!\!\!\!\!\perp_{\mathcal{P}} X_k | S$. We refer to such sets as separating sets. The triple $(X_i, X_j, X_k)$ is labelled as *unambiguous* if at least one such separating set is found and either $X_j$ is in all separating sets or in none of them; otherwise it is labelled as *ambiguous*. If the triple is unambiguous, it is oriented as $v$-structure if and only if $X_j$ is in none of the separating sets. Moreover, in the $v$-structure recovery phase of the PC-like algorithm (Algorithm 9, lines 11-15), the orientation rules are adapted so that only unambiguous triples are oriented. The output of the CPC-like algorithm is a mixed graph in which ambiguous triples are marked. We refer to the combination of the stable PC-like and CPC-like algorithms as the **stable CPC-like algorithm**.

In the case of DAGs, Colombo and Maathuis (Colombo and Maathuis, 2014) found that the CPC-algorithm can be very conservative, in the sense that very few unshielded triples are unambiguous in the sample version, where conditional independence relationships have to be estimated from data. They proposed a minor modification of the CPC approach, called **Majority rule PC algorithm (MPC)** to mitigate the (unnecessary) severity of CPC-like approach. We similarly propose the **Majority rule PC-like algorithm (MPC-like)** for MVR CGs. As in the CPC-like algorithm, we first determine all subsets $S$ of $ad_H(X_i)$ and of $ad_H(X_k)$ that make $X_i$ and $X_k$ conditionally independent, i.e., that satisfy $X_i \perp \!\!\!\!\!\perp_{\mathcal{P}} X_k | S$. The triple $(X_i, X_j, X_k)$ is labelled as *(α, β)-unambiguous* if at least one such separating set
is found or $X_j$ is in no more than $\alpha\%$ or no less than $\beta\%$ of the separating sets, for $0 \leq \alpha \leq \beta \leq 100$. Otherwise it is labelled as ambiguous. (As an example, consider $\alpha = 30$ and $\beta = 60$.) If a triple is unambiguous, it is oriented as a $v$-structure if and only if $X_j$ is in less than $\alpha\%$ of the separating sets. As in the CPC-like algorithm, the orientation rules in the $v$-structure recovery phase of the PC-like algorithm (Algorithm 9, lines 11-15) are adapted so that only unambiguous triples are oriented, and the output is a mixed graph in which ambiguous triples are marked. Note that the CPC-like algorithm is the special case of the MPC-like algorithm with $\alpha = 0$ and $\beta = 100$. We refer to the combination of the stable PC-like and MPC-like algorithms as the stable MPC-like algorithm.

**Theorem 3.28.** Let the distribution of $V$ be faithful to an MVR CG $G$, and assume that we are given perfect conditional independence information about all pairs of variables $(u, v)$ in $V$ given subsets $S \subseteq V \setminus \{u, v\}$. Then the output of the (stable) CPC/MPC-like algorithm is an MVR CG that is Markov equivalent with $G$ that has exactly the minimum set of bidirected edges for its equivalence class.

**Proof.** The skeleton of the learned CG is correct by Theorem 3.26. Now, we prove that for any unshielded triple $(X_i, X_j, X_k)$ in an MVR CG $G$, $X_j$ is either in all sets that $m$-separate $X_i$ and $X_k$ or in none of them. Since $X_i, X_k$ are not adjacent and any MVR chain graph is a maximal ancestral graph (Javidian and Valtorta, 2018c), they are $m$-separated given some subset $S \setminus \{X_i, X_k\}$ due to the maximal property. Based on the pathwise $m$-separation criterion for MVR CGs (see section 3.1), $X_j$ is a collider node in $G$ if and only if $X_j \notin An(S)$. So, $X_j \notin S$. On the other hand, if $X_j$ is a non-collider node then $X_j \in S$, for all $S$ that $m$-separate $X_i$ and $X_k$. Because in this case, $X_j \in An(X_i \cup X_k \cup S)$ and so there is an undirected path $X_i \rightarrow X_j \rightarrow X_k$ in $(G_{An(X_i \cup X_k \cup S)})^a$. Any set $S \setminus \{X_i, X_k\}$ that does not contain $X_j$ will fail to $m$-separate $X_i$ and $X_k$ because of this undirected path. As a result, unshielded triples are all unambiguous. Since all unshielded triples are unambiguous, the orientation rules are as in the (stable) PC-like algorithm. Therefore, the output of the (stable) CPC/MPC-like algorithm is an MVR CG that is Markov equivalent with $G$ that has
exactly the minimum set of bidirected edges for its equivalence class, and soundness and completeness of these rules follows from Sonntag and Peña (Sonntag and Peña, 2012).

**Theorem 3.29.** The decisions about v-structures in the sample version of the stable CPC/MPC-like algorithm is order-independent.

**Proof.** The stable CPC/MPC-like algorithm have order-independent skeleton, by Theorem 3.27. In particular, this means that their unshielded triples and adjacency sets are order-independent. The decision about whether an unshielded triple is unambiguous and/or a v-structure is based on the adjacency sets of nodes in the triple, which are order independent.

**Example 12** (Order-independent decisions about v-structures). We consider the sample versions of the stable CPC/MPC-like algorithm, using the same input as in Example 9. In particular, we assume that all conditional independencies induced by the MVR CG in Figure 3.11(a) are judged to hold except $c \perp \perp d$. Suppose that $c \perp \perp d|b$ and $c \perp \perp d|e$ are thought to hold. Let $\alpha = \beta = 50$.

Denote the skeleton after the skeleton recovery phase by $H$. We consider the unshielded triple $(c, e, d)$. First, we compute $a_H(c) = \{a, d, e\}$ and $a_H(d) = \{a, b, c, e\}$, when $i = 1$. We now consider all subsets $S$ of these adjacency sets, and check whether $c \perp \perp d|S$. The following separating sets are found: $\{b\}$, $\{e\}$, and $\{b, e\}$. Since $e$ is in some but not all of these separating sets, the stable CPC-like algorithm determines that the triple is ambiguous, and no orientations are performed. Since $e$ is in more than half of the separating sets, stable MPC-like determines that the triple is unambiguous and not a v-structure. The output of both algorithms is given in Figure 3.11(c).

At this point it should be clear why the modified PC-like algorithm is labeled “conservative”: it is more cautious than the (stable) PC-like algorithm in drawing unambiguous conclusions about orientations. As we showed in Example 12, the output of the (stable)
CPC-like algorithm may not be collider equivalent with the true MVR CG $G$, if the resulting CG contains an ambiguous triple.

**Order-Independent Orientation Rules**

Even when the skeleton and the determination of the $\nu$-structures are order-independent, Example 10 showed that there might be some order-dependent steps left in the sample version. Regarding the orientation rules, we note that the PC-like algorithm does not suffer from conflicting $\nu$-structures (as shown in (Colombo and Maathuis, 2014) for the PC-algorithm in the case of DAGs), because bi-directed edges are allowed. However, the three orientation rules still suffer from order-dependence issues (see Example 10 and Figure 3.12). To solve this problem, we can use lists of candidate edges for each orientation rule as follows: we first generate a list of all edges that can be oriented by rule R1. We orient all these edges, creating bi-directed edges if there are conflicts. We do the same for rules R2 and R3, and iterate this procedure until no more edges can be oriented.

When using this procedure, we add the letter $L$ (standing for lists), e.g., (stable) LCPC-like and (stable) LMPC-like. The (stable) LCPC-like and (stable) LMPC-like algorithms are fully order-independent in the sample versions. The procedure is illustrated in Example 13.

**Theorem 3.30.** Let the distribution of $V$ be faithful to an MVR CG $G$, and assume that we are given perfect conditional independence information about all pairs of variables $(u, v)$ in $V$ given subsets $S \subseteq V \setminus \{u, v\}$. Then the output of the (stable) LCPC/LMPC-like algorithm is an MVR CG that is Markov equivalent with $G$ that has exactly the minimum set of bidirected edges for its equivalence class.

*Proof.* By Theorem 3.28, we know that the (stable) CPC-like and (stable) MPC-like algorithms are correct. With perfect conditional independence information, there are no conflicts between orientation rules in the essential graph recovery phase of the algorithms.
Therefore, the (stable) LCPC-like and (stable) LMPC-like algorithms are identical to the (stable) CPC-like and (stable) MPC-like algorithms.

\[\]  

**Theorem 3.31.** *The sample versions of stable CPC-like and stable MPC-like algorithms are fully order-independent.*

*Proof.* This follows straightforwardly from Theorems 3.27 and 3.29 and the procedure with lists and bi-directed edges discussed above.

Table 3.3 summarizes the three order-dependence issues explained above and the corresponding modifications of the PC-like algorithm that removes the given order-dependence problem.

Table 3.3: Order-dependence issues and corresponding modifications of the PC-like algorithm that remove the problem. “Yes” indicates that the corresponding aspect of the graph is estimated order-independently in the sample version.

<table>
<thead>
<tr>
<th>skeleton</th>
<th>(v)-structures decisions</th>
<th>edges orientations</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC-like</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>stable PC-like</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>stable CPC/MPC-like</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>stable LCPC/LMPC-like</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Example 13.** Consider the structure shown in Figure 3.12. As a first step, we construct a list containing all candidate structures eligible for orientation rule R1 in the phase of the essential graph recovery. The list contains the unshielded triples \((e, c, d), (c, d, f), (a, c, d),\) and \((b, d, c)\). Now, we go through each element in the list and we orient the edges accordingly, allowing bi-directed edges. This yields the edge orientation \(c \leftrightarrow d\), regardless of the ordering of the variables.

3.4.3 **EVALUATION**

In this subsection, we compare the performance of our algorithms (Table 3.3) with the original PC-like learning algorithm by running them on randomly generated MVR chain
graphs in low-dimensional and high-dimensional data, respectively. We report on the Gaussian case only because of space limitations.

**Performance Evaluation on Random MVR CGs (Gaussian case)**

To investigate the performance of the proposed learning methods in this paper, we use the same approach that Ma, Xie, and Geng (2008) used in evaluating the performance of the LCD algorithm on LWF chain graphs. We run our algorithms on randomly generated MVR chain graphs and then we compare the results and report summary error measures in all cases.

**Data Generation Procedure.** First we explain the way in which the random MVR chain graphs and random samples are generated. Given a vertex set \( V \), let \( p = |V| \) and \( N \) denote the average degree of edges (including bidirected, pointing out, and pointing in) for each vertex. We generate a random MVR chain graph on \( V \) as follows:

- Order the \( p \) vertices and initialize a \( p \times p \) adjacency matrix \( A \) with zeros;
- Set each element in the lower triangle part of \( A \) to be a random number generated from a Bernoulli distribution with probability of occurrence \( s = N/(p - 1) \);
- Symmetrize \( A \) according to its lower triangle;
- Select an integer \( k \) randomly from \( \{1, \ldots, p\} \) as the number of chain components;
- Split the interval \([1, p]\) into \( k \) equal-length subintervals \( I_1, \ldots, I_k \) so that the set of variables into each subinterval \( I_m \) forms a chain component \( C_m \);
- Set \( A_{ij} = 0 \) for any \((i, j)\) pair such that \( i \in I_l, j \in I_m \) with \( l > m \).

This procedure yields an adjacency matrix \( A \) for a chain graph with \((A_{ij} = A_{ji} = 1)\) representing a bidirected edge between \( V_i \) and \( V_j \) and \((A_{ij} = 1, A_{ji} = 0)\) representing a directed edge from \( V_i \) to \( V_j \). Moreover, it is not difficult to see that \( \mathbb{E}[\text{vertex degree}] = N \), where an adjacent vertex can be linked by either a bidirected or a directed edge. In order
to sample the artificial CGs, we first transform them into DAGs and then generate samples from these DAGs under marginalization, as indicated in (Javidian and Valtorta, 2018c), using Hugin.

**Experimental Results.** We evaluate the performance of the proposed algorithms in terms of the six measurements that are commonly used for constraint-based learning algorithms: (a) the true positive rate (TPR) (also known as sensitivity, recall, and hit rate), (b) the false positive rate (FPR) (also known as fall-out), (c) the true discovery rate (TDR) (also known as precision or positive predictive value), (d) accuracy (ACC) for the skeleton, (e) the structural Hamming distance (SHD) (this is the metric described in (Tsamardinos et al., 2006) to compare the structure of the learned and the original graphs), and (f) run-time for the LCG recovery algorithms. In short, $TPR = \frac{true\ positive\ (TP)}{the\ number\ of\ positive\ cases\ in\ the\ data\ (Pos)}$ is the ratio of the number of correctly identified edges over total number of edges (in true graph), $FPR = \frac{false\ positive\ (FP)}{the\ number\ of\ negative\ cases\ in\ the\ data\ (Neg)}$ is the ratio of the number of incorrectly identified edges over total number of gaps, $TDR = \frac{true\ positive\ (TP)}{the\ total\ number\ of\ edges\ in\ the\ recovered\ CG}$ is the ratio of the number of correctly identified edges over total number of edges (both in estimated graph), $ACC = \frac{true\ positive\ (TP)+\ true\ negative\ (TN)}{Pos+Neg}$ and $SHD$ is the number of legitimate operations needed to change the current resulting graph to the true essential graph, where legitimate operations are: (a) add or delete an edge and (b) insert, delete or reverse an edge orientation.

In principle, large values of TPR, TDR, and ACC, and small values of FPR and SHD indicate good performance. All of these six measurements are computed on the essential graphs of the CGs, rather than the CGs directly, to avoid spurious differences due to random orientation of undirected edges.

In our simulation, for low-dimensional settings, we set $N$ (expected number of adjacent vertices) to 2 and change the parameters $p$ (the number of vertices) and $n$ (sample size) as follows:

- $p \in \{10, 20, 30, 40, 50\}$,
\[ n \in \{500, 1000, 5000, 10000\}. \]

For each \((p, N)\) combination, we first generate 30 random MVR CGs. We then generate a random Gaussian distribution based on each graph (transformed DAG) and draw an identically independently distributed (i.i.d.) sample of size \(n\) from this distribution for each possible \(n\). For each sample, four different significance levels \((\alpha = 0.001, 0.005, 0.01, 0.05)\) are used to perform the hypothesis tests. The null hypothesis \(H_0\) is “two variables \(u\) and \(v\) are conditionally independent given a set \(C\) of variables” and alternative \(H_1\) is that \(H_0\) may not hold. We then compare the results to access the influence of the significance testing level on the performance of our algorithms.

For the high-dimensional setting, we generate 30 random MVR CGs with 1000 vertices for which the expected number of adjacent vertices for each vertex is 2. We then generate a random Gaussian distribution based on each graph (transformed DAG) and draw an identically independently distributed (i.i.d.) sample of size 50 from this distribution for each DAG. These numbers are similar to ones that could be encountered in gene regulatory network experiments (Colombo and Maathuis, 2014, section 6).

Figure 3.13 shows that: (a) as we expected Ma, Xie, and Geng (2008); Kalisch and Bühlmann (2007), all algorithms work well on sparse graphs \((N = 2)\), (b) for all algorithms, typically the TPR, TDR, and ACC increase with sample size, (c) for all algorithms, typically the SHD and FPR decrease with sample size, (d) a large significance level \((\alpha = 0.05)\) typically yields large TPR, FPR, and SHD, (e) while the stable PC-like algorithm has a better TDR and FPR in comparison with the original PC-like algorithm, the original PC-like algorithm has a better TPR (as observed in the case of DAGs Colombo and Maathuis (2014)). This can be explained by the fact that the stable PC-like algorithm tends to perform more tests than the original PC-like algorithm, and (h) while the original PC-like algorithm has a (slightly) better SHD in comparison with the stable PC-like algorithm in low-dimensional data, the stable PC-like algorithm has a better SHD in high-dimensional data. Also, (very) small variances indicate that the order-independent versions of the PC-
Figure 3.13: The first two rows show the performance of the original (OPC) and stable PC-like (SPC) algorithms for randomly generated Gaussian chain graph models: average over 30 repetitions with 50 variables correspond to N = 2, and the significance level \( \alpha = 0.001 \). The last two rows show the performance of the original (OPC) and stable PC-like (SPC) algorithms for randomly generated Gaussian chain graph models: average over 30 repetitions with 1000 variables correspond to N = 2, sample size S = 50, and the significance level \( \alpha = 0.05, 0.01, 0.005, 0.001 \).
like algorithm in high-dimensional data are stable. When considering average running
times versus sample sizes, as shown in Figure 3.13, we observe that: (a) the average run
time increases when sample size increases; (b) generally, the average run time for the origi-
nal PC-like algorithm is (slightly) better than that for the stable PC-like algorithm in both
low and high dimensional settings.

In summary, empirical simulations show that our algorithms achieve competitive results
with the original PC-like learning algorithm; in particular, in the Gaussian case the order-
independent algorithms achieve output of better quality than the original PC-like algorithm,
especially in high-dimensional settings. Algorithm 9 and the stable PC-like algorithms
have been implemented in the R language (https://github.com/majavid/SUM2019).

3.5 A Decomposition-Based Algorithm for Structure Learning of MVR Chain Graphs

MVR chain graphs were introduced by Cox and Wermuth in the nineties to combine
directed acyclic graphs (representing the structure of Bayesian networks) and bidirected graphs (covariance graphs). As mentioned earlier, one important aspect of PGMs in general, and chain graphs especially, is the possibility of learning the structure of models directly from sampled data. Two constraint-based learning algorithms, that use a statistical analysis to test the presence of a conditional independency, exist for learning MVR CGs: (1) the PC-like algorithm (Sonntag and Peña, 2012), and (2) the answer set programming (ASP) algorithm (Peña, 2016b).

In this section, we extend the decomposition approach for learning Bayesian networks (BNs) proposed by (Xie, Zheng, and Zhao, 2006) to learning multivariate regression chain graphs (MVR CGs), which include BNs as a special case. The same advantages of this decomposition approach hold in the more general setting: reduces complexity and increased power of computational independence tests. Moreover, latent (hidden) variables can be represented in MVR CGs by using bidirected edges, and our algorithm correctly recovers any independence structure that is faithful to an MVR CG, thus greatly extending the range of applications of decomposition-based model selection techniques. Simulations under a variety of settings demonstrate the competitive performance of our method in comparison with the PC-like algorithm (Sonntag and Peña, 2012). In fact, the decomposition-based algorithm usually outperforms the PC-like algorithm except in running time. The performance of both algorithms is much better when the underlying graph is sparse.

3.5.1 Construction of Undirected Independence Graphs and m-Separation Trees

In this section we consider graphs containing both directed (→) and bidirected (↔) edges and largely use the terminology of (Xie, Zheng, and Zhao, 2006; Richardson, 2003), where the reader can also find further details. Below we briefly list some of the most central concepts used in this section.

Let $\tilde{G}_V = (V, \tilde{E}_V)$ denote an undirected graph where $\tilde{E}_V$ is a set of undirected edges. An undirected edge between two vertices $u$ and $v$ is denoted by $(u, v)$. For a subset $A$ of $V$, let
\( \bar{G}_A = (A, \bar{E}_A) \) be the subgraph induced by \( A \) and \( \bar{E}_A = \{ e \in \bar{E}_V | e \in A \times A \} = \bar{E}_V \cap (A \times A) \).

An undirected graph is called complete if any pair of vertices is connected by an edge. For an undirected graph, we say that vertices \( u \) and \( v \) are separated by a set of vertices \( Z \) if each path between \( u \) and \( v \) passes through \( Z \). We say that two distinct vertex sets \( X \) and \( Y \) are separated by \( Z \) if and only if \( Z \) separates every pair of vertices \( u \) and \( v \) for any \( u \in X \) and \( v \in Y \). We say that an undirected graph \( \bar{G}_V \) is an undirected independence graph (UIG) for \( CG \) \( G \) if the fact that a set \( Z \) separates \( X \) and \( Y \) in \( \bar{G}_V \) implies that \( Z \) \( m \)-separates \( X \) and \( Y \) in \( G \). Note that the augmented graph derived from \( CG \) \( G \), \( (G)_a \), is an undirected independence graph for \( G \). We say that \( \bar{G}_V \) can be decomposed into subgraphs \( \bar{G}_A \) and \( \bar{G}_B \) if

1. \( A \cup B = V \), and

2. \( C = A \cap B \) separates \( V \setminus A \) and \( V \setminus B \) in \( \bar{G}_V \).

The above decomposition does not require that the separator \( C \) be complete, which is required for weak decomposition defined in (Lauritzen, 1996). We show that a problem of structural learning of \( CG \) can also be decomposed into problems for its decomposed subgraphs even if the separator is not complete.

A triangulated (chordal) graph is an undirected graph in which all cycles of four or more vertices have a chord, which is an edge that is not part of the cycle but connects two vertices of the cycle (see, for example, Figure 3.15). For an undirected graph \( \bar{G}_V \) which is not triangulated, we can add extra ("fill-in") edges to it such that it becomes to be a triangulated graph, denoted by \( \bar{G}_V' \).

Let \( X \perp \perp Y \) denote the independence of \( X \) and \( Y \), and \( X \perp \perp Y | Z \) (or \( \langle X, Y | Z \rangle \)) the conditional independence of \( X \) and \( Y \) given \( Z \). We assume that all independencies of a probability distribution of variables in \( V \) can be checked by \( m \)-separations of \( G \), called the faithfulness assumption (Spirtes, Glymour, and Scheines, 2000). The faithfulness assumption means that all independencies and conditional independencies among variables can be represented by \( G \).
The global skeleton is an undirected graph obtained by dropping direction of CG. Note that the absence of an edge \((u, v)\) implies that there is a variable subset \(S\) of \(V\) such that \(u\) and \(v\) are independent conditional on \(S\), that is, \(u \perp \perp v\) \(\mid S\) for some \(S \subseteq V \setminus \{u, v\}\) (Javidian and Valtorta, 2018c). Two MVR CGs over the same variable set are called Markov equivalent if they induce the same conditional independence restrictions. Two MVR CGs are Markov equivalent if and only if they have the same global skeleton and the same set of \(v\)-structures (unshielded colliders) (Wermuth and Sadeghi, 2012). An equivalence class of MVR CGs consists of all MVR CGs which are Markov equivalent, and it is represented as a partially directed graph (i.e., a graph containing directed, undirected, and bidirected edges and no directed cycles) where the directed/bidirected edges represent edges that are common to every MVR CG in it, while the undirected edges represent that any appropriate orientation of them leads to a Markov equivalent MVR CG. Therefore the goal of structural learning is to construct a partially directed graph to represent the equivalence class. A local skeleton for a subset \(A\) of variables is an undirected subgraph for \(A\) in which the absence of an edge \((u, v)\) implies that there is a subset \(S\) of \(A\) such that \(u \perp \perp v\mid S\).

Now, we introduce the notion of \(m\)-separation trees, which is used to facilitate the representation of the decomposition. The concept is similar to the junction tree of cliques and the independence tree introduced for DAGs as \(d\)-separation trees in (Xie, Zheng, and Zhao, 2006). Let \(C = \{C_1, \ldots, C_H\}\) be a collection of distinct variable sets such that for \(h = 1, \ldots, H, C_h \subseteq V\). Let \(T\) be a tree where each node corresponds to a distinct variable.
set in $C$, to be displayed as an oval (see, for example, Figure 3.16). The term ‘node’ is used for an $m$-separation tree to distinguish from the term ‘vertex’ for a graph in general. An undirected edge $e = (C_i, C_j)$ connecting nodes $C_i$ and $C_j$ in $T$ is labeled with a separator $S = C_i \cap C_j$, which is displayed as a rectangle. Removing an edge $e$ or equivalently, removing a separator $S$ from $T$ splits $T$ into two subtrees $T_1$ and $T_2$ with node sets $C_1$ and $C_2$ respectively. We use $V_i$ to denote the union of the vertices contained in the nodes of the subtree $T_i$ for $i = 1, 2$.

**Definition 3.32.** A tree $T$ with node set $C$ is said to be an $m$-separation tree for chain graph $G = (V, E)$ if

- $\cup_{C_i \in C} C_i = V$, and

- for any separator $S$ in $T$ with $V_1$ and $V_2$ defined as above by removing $S$, we have $\langle V_1 \setminus S, V_2 \setminus S \setminus S \rangle_G$.

![Figure 3.16: An $m$-separation tree.](image)

Notice that a separator is defined in terms of a tree whose nodes consist of variable sets, while the $m$-separator is defined based on chain graph. In general, these two concepts are not related, though for an $m$-separation tree its separator must be some corresponding $m$-separator in the underlying MVR chain graph. The definition of $m$-separation trees for MVR chain graphs is similar to that of junction trees of cliques, see (Cowell et al., 1999; Lauritzen, 1996). Actually, it is not difficult to see that a junction tree of chain graph $G$ is also an $m$-separation tree. However, as in (Ma, Xie, and Geng, 2008), we point out
two differences here: (a) an $m$-separation tree is defined with $m$-separation and it does not require that every node is a clique or that every separator is complete on the augmented graph; (b) junction trees are mostly used as inference engines, while our interest in $m$-separation trees is mainly derived from their power in facilitating the decomposition of structural learning.

A collection of variable sets $C = \{C_1, \ldots, C_H\}$ is said to be a hypergraph on $V$ where each hyperedge $C_h$ is a nonempty subset of variables, and $\bigcup_{h=1}^{H} C_h = V$. A hypergraph is a reduced hypergraph if $C_i \not\subseteq C_j$ for $i \neq j$. In this section, only reduced hypergraphs are used, and thus simply called hypergraphs.

As proposed in (Xie, Zheng, and Zhao, 2006), one can construct a $d$-separation tree from observed data, from domain or prior knowledge of conditional independence relations or from a collection of databases. However, their arguments are not valid for constructing an $m$-separation tree from domain knowledge or from observed data patterns in the current setting. In this subsection, we first extend Theorem 2 of (Xie, Zheng, and Zhao, 2006), which guarantees that their method for constructing a separation tree from data is valid for MVR chain graphs. Then we investigate sufficient conditions for constructing $m$-separation trees from domain or prior knowledge of conditional independence relations or from a collection of databases.

**Constructing an $m$-Separation Tree from Observed Data**

In several algorithms for structural learning of PGMs, the first step is to construct an undirected independence graph in which the absence of an edge $(u, v)$ implies $u \perp \perp v | V \setminus \{u, v\}$. To construct such an undirected graph, we can start with a complete undirected graph, and then for each pair of variables $u$ and $v$, an undirected edge $(u, v)$ is removed if $u$ and $v$ are independent conditional on the set of all other variables (Xie, Zheng, and Zhao, 2006). For normally distributed data, the undirected independence graph can be efficiently constructed by removing an edge $(u, v)$ if and only if the corresponding entry in the concentration ma-
trix (inverse covariance matrix) is zero (Lauritzen, 1996, Proposition 5.2). For this purpose, performing a conditional independence test for each pair of random variables using the partial correlation coefficient can be used. If the $p$-value of the test is smaller than the given threshold, then there will be an edge on the output graph. For discrete data, a test of conditional independence given a large number of discrete variables may be of extremely low power. To cope with such difficulty, a local discovery algorithm called Max-Min Parents and Children (MMPC) (Tsamardinos, Aliferis, and Statnikov, 2003) or the forward selection procedure described in (Edwards, 2000) can be applied.

An $m$-separation tree can be built by constructing a junction tree (Jensen and Nielsen, 2007) from an undirected independence graph. In fact, we generalize Theorem 2 of (Xie, Zheng, and Zhao, 2006) as follows.

**Theorem 3.33.** A junction tree constructed from an undirected independence graph for MVR CG $G$ is an $m$-separation tree for $G$.

An $m$-separation tree $T$ only requires that all $m$-separation properties of $T$ also hold for MVR CG $G$, but the reverse is not required. Thus we only need to construct an undirected independence graph that may have fewer conditional independencies than the moral graph, and this means that the undirected independence graph may have extra edges added to the augmented graph. As (Xie, Zheng, and Zhao, 2006) observe for $d$-separation in DAGs, if all nodes of an $m$-separation tree contain only a few variables, “the null hypothesis of the absence of an undirected edge may be tested statistically at a larger significance level.”

Since there are standard algorithms for constructing junction trees from UIGs (Cowell et al., 1999, Chapter 4, Section 4), the construction of separation trees reduces to the construction of UIGs. In this sense, Theorem 3.33 enables us to exploit various techniques for learning UIGs to serve our purpose. More suggested methods for learning UIGs from data, in addition to the above mentioned techniques, can be found in (Ma, Xie, and Geng, 2008).

**Example 14.** To construct an $m$-separation tree for MVR CG $G$ in Figure 3.15(a), at first
an undirected independence graph is constructed by starting with a complete graph and removing an edge \((u, v)\) if \( u \perp \!\!\!\perp v \mid V \setminus \{u, v\} \). An undirected graph obtained in this way is the augmented graph of MVR CG \( G \). In fact, we only need to construct an undirected independence graph which may have extra edges added to the augmented graph. Next triangulate the undirected graph and finally obtain the \( m \)-separation tree, as shown in Figure 3.15(b) and Figure 3.16 respectively.

**Constructing an \( m \)-Separation Tree from Domain Knowledge or from Observed Data Patterns**

Algorithm 2 of (Xie, Zheng, and Zhao, 2006) proposes an algorithm for constructing a \( d \)-separation tree \( T \) from domain knowledge or from observed data patterns such that a correct skeleton can be constructed by combining subgraphs for nodes of \( T \). In this subsection, we propose an approach for constructing an \( m \)-separation tree from domain knowledge or from observed data patterns without conditional independence tests. Domain knowledge of variable dependencies can be represented as a collection of variable sets \( C = \{C_1, \ldots, C_H\} \), in which variables contained in the same set may associate with each other directly but variables contained in different sets associate with each other through other variables. This means that two variables that are not contained in the same set are independent conditionally on all other variables. On the other hand, in an application study, observed data may have a collection of different observed patterns, \( C = \{C_1, \ldots, C_H\} \), where \( C_h \) is the set of observed variables for the \( h \)th group of individuals. In both cases, the condition to make our algorithms correct for structural learning from a collection \( C \) is that \( C \) must contain sufficient data such that parameters of the underlying MVR CG are estimable.

For a DAG, parameters are estimable if, for each variable \( u \), there is an observed data pattern \( C_h \) in \( C \) that contains both \( u \) and its parent set. Thus a collection \( C \) of observed patterns has sufficient data for correct structural learning if there is a pattern \( C_h \) in \( C \) for each \( u \) such that \( C_h \) contains both \( u \) and its parent set in the underlying DAG. Also, domain knowledge is legitimate if, for each variable \( u \), there is a hyperedge \( C_h \) in \( C \) that contains
both \( u \) and its parent set (Xie, Zheng, and Zhao, 2006). However, these conditions are not valid in the case of MVR chain graphs. In fact, for MVR CGs domain knowledge is legitimate if for each connected component \( \tau \), there is a hyperedge \( C_h \) in \( C \) that contains both \( \tau \) and its parent set \( pa_G(\tau) \). Also, a collection \( C \) of observed patterns has sufficient data for correct structural learning if there is a pattern \( C_h \) in \( C \) for each connected component \( \tau \) such that \( C_h \) contains both \( \tau \) and its parent set \( pa_G(\tau) \) in the underlying MVR CG.

Algorithm 11: Construct an \( m \)-separation tree from a hypergraph

**Input:** a hypergraph \( C = \{C_1, \ldots, C_H\} \), where each hyperedge \( C_h \) is a variable set such that for each connected component \( \tau \), there is a hyperedge \( C_h \) in \( C \) that contains both \( \tau \) and its parent set \( pa_G(\tau) \).

**Output:** \( T \), which is an \( m \)-separation tree for the hypergraph \( C \).

1. For each hyperedge \( C_h \), construct a complete undirected graph \( \bar{G}_h \) with the edge set \( \bar{E}_h = \{(u, v) | u, v \in C_h\} = C_h \times C_h \);
2. Construct the entire undirected graph \( \bar{G}_V = (V, \bar{E}) \), where \( \bar{E} = \bar{E}_1 \cup \cdots \cup \bar{E}_H \);
3. Construct a junction tree \( T \) by triangulating \( \bar{G}_V \).

The correctness of Algorithm 11 is proven in Appendix C. Note that we do not need any conditional independence test in Algorithm 11 to construct an \( m \)-separation tree. In this algorithm, we can use the proposed algorithm in (Berry et al., 2004) to construct a minimal triangulated graph. In order to illustrate Algorithm 11, see Figure 3.17.

Guaranteeing the presence of both \( \tau \) and its parent set \( pa(\tau) \) in at least one hyperedge, as required in Algorithm 11, is a strong requirement, which may prevent the use of domain knowledge as a practical source of information for constructing MVR chain graphs. In addition, we remark that answering the question "how can one obtain this information?" is beyond the scope of this paper. The two examples that follow show that restricting the hyperedge contents in two natural ways lead to errors.

The example illustrated in Figure 3.18 shows that, if for each variable \( u \) there is a hyperedge \( C_h \) in \( C \) that contains both \( u \) and its parent set, we cannot guarantee the correctness of our algorithm. Note that vertices \( a \) and \( d \) are separated in the tree \( T \) of Figure 3.18 part (d) by removing vertex \( b \), but \( a \) and \( d \) are not \( m \)-separated given \( b \) as can be verified using
Figure 3.17: Construction the $m$-separation tree. (a) An MVR CG. (b) Domain knowledge of associations. (c) The undirected graph and triangulation. (d) The $m$-separation tree $T$.

3.18 part (a).

The example illustrated in Figure 3.19 shows that, if for each variable $u$ there is a hyper-edge $C_h$ in $C$ that contains both $u$ and its boundary set, Algorithm 11 does not necessarily give an $m$-separation tree because, for example, $S = \{a, b\}$ separates $c$ and $d$ in tree $T$ of Figure 3.19 part (d), but $S$ does not $m$-separate $c$ and $d$ in the MVR CG $G$ in Figure 3.19 part (a).

3.5.2 Decomposition of Structural Learning

Applying the following theorem to structural learning, we can split a problem of searching for $m$–separators and building the skeleton of CG into small problems for every node of $m$-separation tree $T$.

**Theorem 3.34.** Let $T$ be an $m$-separation tree for CG $G$. Vertices $u$ and $v$ are $m$-separated by $S \subseteq V$ in $G$ if and only if (i) $u$ and $v$ are not contained together in any node $C$ of $T$ or (ii) there exists a node $C$ that contains both $u$ and $v$ such that a subset $S'$ of $C$, $m$-separates $u$ and $v$. 

102
According to Theorem 3.34, a problem of searching for an $m$-separator $S$ of $u$ and $v$ in all possible subsets of $V$ is localized to all possible subsets of nodes in an $m$-separation tree that contain $u$ and $v$. For a given $m$-separation tree $T$ with the node set $C = \{C_1, \ldots, C_H\}$, we can recover the skeleton and all $v$-structures for CG as follows. First we construct a local skeleton for every node $C_h$ of $T$, which is constructed by starting with a complete undirected subgraph and removing an undirected edge $(u, v)$ if there is a subset $S$ of $C_h$ such that $u$ and $v$ are independent conditional on $S$. Then, in order to construct the global skeleton, we combine all these local skeletons together and remove edges that are present in some local skeletons but absent in other local skeletons. Then we determine every $v$-structure if two non-adjacent vertices $u$ and $v$ have a common neighbor in the global skeleton but the neighbor is not contained in the $m$-separator of $u$ and $v$. Finally we can orient more undirected edges if none of them creates either a partially directed cycle or a
Figure 3.19: Insufficiency of having a hypergraph that contains both $u$ and its boundary set for every $u \in V$. (a) An MVR CG. (b) Domain knowledge of associations. (c) The undirected graph constructed by union of complete graphs corresponding to each hyperedge, which is also a triangulated graph. (d) The junction tree $T$, which is not an $m$-separation tree.

new $\nu$-structure (see, for example, Figure 3.20). This process is formally described in the following algorithm:

The following algorithm returns an MVR chain graph that contains exactly the minimum set of bidirected edges for its Markov equivalence class. For the correctness of lines 2-7 in Algorithm 13, see (Sonntag and Peña, 2012).

Figure 3.20: (a) Local skeletons for every node of $T$. (b) The global skeleton and all $\nu$-structures.

The following algorithm returns an MVR chain graph that contains exactly the minimum set of bidirected edges for its Markov equivalence class. For the correctness of lines 2-7 in Algorithm 13, see (Sonntag and Peña, 2012).

Figure 3.21: The Rules (Sonntag and Peña, 2012)
Algorithm 12: A recovery algorithm for MVR chain graphs

Input: a probability distribution $p$ faithful to an unknown MVR CG $G$. 
Output: the pattern of MVR CG $G$.

1. Construct an $m$-separation tree $T$ with a node set $C = \{C_1, \ldots, C_H\}$ as discussed in Section 3.5.1;
2. Set $S = \emptyset$;
3. for $h \leftarrow 1$ to $H$ do 
   
   4. Start from a complete undirected graph $\tilde{G}_h$ with vertex set $C_h$;
   
   5. for each vertex pair $\{u, v\} \subseteq C_h$ do 
      
      6. if $\exists S_{uv} \subseteq C_h$ such that $u \perp \perp v \mid S_{uv}$ then
         
         7. Delete the edge $(u, v)$ in $\tilde{G}_h$;
         
         8. Add $S_{uv}$ to $S$;
      
      end
   
   end
3. for each $S_{uv}$ in the list $S$ do 
   
   4. if $u \circ \neg w \neg v$ appears in the global skeleton and $w$ is not in $S_{uv}$ then
      
      /* $u \circ \neg w$ means $u \leftarrow w$ or $u \Rightarrow w$. Also, $w \circ \neg v$ means $w \rightarrow v$ or $w \Leftarrow v$. */
      
      5. Determine a $v$-structure $u \circ \rightarrow w \Leftarrow \neg v$;
   
   end

According to Theorem 3.34, we can prove that the global skeleton and all $v$-structures obtained by applying the decomposition in Algorithm 12 are correct, that is, they are the same as those obtained from the joint distribution of $V$, see Appendix C for the details of proof.

Note that separators in an $m$-separation tree may not be complete in the augmented graph. Thus the decomposition is weaker than the decomposition usually defined for parameter estimation (Cowell et al., 1999; Lauritzen, 1996).
Algorithm 13: A recovery algorithm for MVR chain graphs with minimum set of bidirected edges for its equivalence class

**Input:** a probability distribution $p$ faithful to an unknown MVR CG $G$.

**Output:** an MVR CG $G'$ s.t. $G$ and $G'$ are Markov equivalent and $G'$ has exactly the minimum set of bidirected edges for its equivalence class.

1. Call Algorithm 12 to construct $G'$, which is the equivalence class of MVR CGs for $G$;
2. Apply rules 1-3 in Figure 3.21 while possible;
   /* After this line, the learned graph is the essential graph of MVR CG $G$ i.e., it has the same skeleton as $G$ and contain all and only the arrowheads that are shared by all MVR CGs in the Markov equivalence class of $G$ (Sonntag, Peña, and Gómez-Olmedo, 2015). */
3. Let $G'_u$ be the subgraph of $G'$ containing only the nodes and the undirected edges in $G'$;
4. Let $T$ be the junction tree of $G'_u$;
   /* If $G'_u$ is disconnected, the cliques belonging to different connected components can be linked with empty separators, as described in (Golumbic, 1980, Theorem 4.8). */
5. Order the cliques $C_1, \ldots, C_n$ of $G'_u$ s.t. $C_1$ is the root of $T$ and if $C_i$ is closer to the root than $C_j$ in $T$ then $C_i < C_j$;
6. Order the nodes such that if $A \in C_i$, $B \in C_j$, and $C_i < C_j$ then $A < B$;
7. Orient the undirected edges in $G'$ according to the ordering obtained in line 6.

3.5.3 Complexity Analysis and Advantages

In this subsection, we start by comparing our algorithm with the main algorithm in (Xie, Zheng, and Zhao, 2006) that is designed specifically for DAG structural learning when the underlying graph structure is a DAG. We make this choice of the DAG specific algorithm so that both algorithms can have the same separation tree as input and hence are directly comparable.

In a DAG, all chain components are singletons. Therefore, sufficiency of having a hypergraph that contains both $\tau$ and its parent set for every chain component is equivalent with having a hypergraph that contains both $u$ and its parent set for every $u \in V$, when the underlying graph structure is a DAG. Therefore, it is obvious that our algorithm has the same effect and the same complexity as the main algorithm in (Xie, Zheng, and Zhao, 2006).
The same advantages mentioned by (Xie, Zheng, and Zhao, 2006) for their BN structural learning algorithm hold for our algorithm when applied to MVR CGs. For the reader convenience, we list them here. First, by using the \( m \)-separation tree, independence tests are performed only conditionally on smaller sets contained in a node of the \( m \)-separation tree rather than on the full set of all other variables. Thus our algorithm has higher power for statistical tests. Second, the computational complexity can be reduced. This complexity analysis focuses only on the number of conditional independence tests for constructing the equivalence class. Decomposition of graphs is a computationally simple task compared to the task of testing conditional independence for a large number of triples of sets of variables. The triangulation of an undirected graph is used in our algorithms to construct an \( m \)-separation from an undirected independence graph. Although the problem for optimally triangulating an undirected graph is NP-hard, sub-optimal triangulation methods (Berry et al., 2004) may be used provided that the obtained tree does not contain too large nodes to test conditional independencies. Two of the best known algorithms are lexicographic search and maximum cardinality search, and their complexities are \( O(|V||E|) \) and \( O(|V| + |E|) \), respectively (Berry et al., 2004). Thus in our algorithms, the conditional independence tests dominate the algorithmic complexity.

The complexity of the Algorithm 12 is \( O(Hm^22^m) \) as claimed in (Xie, Zheng, and Zhao, 2006, Section 6), where \( H \) is the number of hyperedges (usually \( H \ll |V| \)) and \( m = \max_h |C_h| \) where \( |C_h| \) denotes the number of variables in \( C_h \) (\( m \) usually is much less than \( |V| \)).

### 3.5.4 Evaluation

In this subsection, we evaluate the performance of our algorithms in various setups using simulated / synthetic data sets. We first compare the performance of our algorithm with the PC-like learning algorithm (Sonntag and Peña, 2012) by running them on randomly generated MVR chain graphs. (A brief description of the PC-like algorithm is provided at
the beginning of section 3.6.) We then compare our method with the PC-like algorithm on
different discrete Bayesian networks such as ASIA, INSURANCE, ALARM, and HAIL-
FINDER that have been widely used in evaluating the performance of structural learning
algorithms. Empirical simulations show that our algorithm achieves competitive results
with the PC-like learning algorithm; in particular, in the Gaussian case the decomposition-
based algorithm outperforms (except in running time) the PC-like algorithm. Algorithms
12, 13, and the PC-like algorithm have been implemented in the R language. All the results
reported here are based on our R implementation (Javidian and Valtorta, 2019b).

**Performance Evaluation on Random MVR Chain Graphs (Gaussian case)**

To investigate the performance of the decomposition-based learning method, we use the
same approach that (Ma, Xie, and Geng, 2008) used in evaluating the performance of the
LCD algorithm on LWF chain graphs. We run our algorithms and the PC-like algorithm
on randomly generated MVR chain graphs and then we compare the results and report
summary error measures in all cases.

**Data Generation Procedure.** First we explain the way in which the random MVR chain
graphs and random samples are generated. Given a vertex set $V$, let $p = |V|$ and $N$ denote
the average degree of edges (including bidirected and pointing out and pointing in) for each
vertex.

We generate a random MVR chain graph on $V$ as follows:

- Choose one element, say $k$, of the vector $c = (0.1, 0.2, 0.3, 0.4, 0.5)$ randomly$^4$.

- Use the randDAG function from the pcalg R package and generate an un-weighted
  random Erdos-Renyi graph, which is a DAG with $p + (k \times p)$ nodes and $N$ expected
  number of neighbours per node.

$^4$In the case of $p = 40, 50$ we use $c = (0.1, 0.2)$. 

• Use the AG function from the ggm R package and marginalize out $k \times p$ nodes to obtain a random MVR chain graph with $p$ nodes and $N$ expected number of neighbours per node. If the obtained graph is not an MVR chain graph, repeat this procedure until an MVR CG is obtained.

The rnorm.cg function from the lcd R package was used to generate a desired number of normal random samples from the canonical DAG (Richardson and Spirtes, 2002) corresponding to the obtained MVR chain graph in the first step. Notice that faithfulness is not necessarily guaranteed by the current sampling procedure (Ma, Xie, and Geng, 2008).

Experimental Results for Random MVR Chain Graphs (Gaussian case). We evaluate the performance of the decomposition-based and PC-like algorithms in terms of five measurements: (a) the true positive rate (TPR)$^5$, (b) the false positive rate (FPR)$^6$, (c) accuracy (ACC) for the skeleton, (d) the structural Hamming distance (SHD)$^7$, and (e) run-time for the pattern recovery algorithms. In short, $TPR = \frac{\text{true positive (TP)}}{\text{the number of real positive cases in the data (Pos)}}$ is the ratio of the number of correctly identified edges over total number of edges, $FPR = \frac{\text{false positive (FP)}}{\text{the number of real negative cases in the data (Neg)}}$ is the ratio of the number of incorrectly identified edges over total number of gaps, $ACC = \frac{\text{true positive (TP)} + \text{true negative (TN)}}{\text{Pos+Neg}}$ and $SHD$ is the number of legitimate operations needed to change the current pattern to the true one, where legitimate operations are: (a) add or delete an edge and (b) insert, delete or reverse an edge orientation. In principle, a large TPR and ACC, a small FPR and SHD indicate good performance.

In our simulation, we change three parameters $p$ (the number of vertices), $n$ (sample size) and $N$ (expected number of adjacent vertices) as follows:

- $p \in \{10, 20, 30, 40, 50\}$,

---

$^5$Also known as sensitivity, recall, and hit rate.

$^6$Also known as fall-out.

$^7$This is the metric described in (Tsamardinos et al., 2006) to compare the structure of the learned and the original graphs.
\[ n \in \{300, 1000, 3000, 10000\}, \text{ and} \]
\[ N \in \{2, 3, 5, 8, 10\}. \]

For each \((p, N)\) combination, we first generate 25 random MVR chain graphs. We then generate a random Gaussian distribution based on each corresponding canonical DAG and draw an identically independently distributed (i.i.d.) sample of size \(n\) from this distribution for each possible \(n\), and finally we remove those columns (if any exist) that correspond to the hidden variables. For each sample, three different significance levels \(\alpha = 0.05/0.01/0.005\) are used to perform the hypothesis tests. For decomposition-based algorithm we consider two different versions: The first version uses Algorithm 12 and the three rules in Algorithm 13, while the second version uses both Algorithm 12 and 13. Since the learned graph of the first version may contain some undirected edges, we call it the essential recovery algorithm. However, removing all directed and bidirected edges from the learned graph results in a chordal graph (Sonntag and Peña, 2012). Furthermore, the learned graph has exactly the (unique) minimum set of bidirected edges for its Markov equivalence class (Sonntag and Peña, 2012). The second version of the decomposition-based algorithm returns an MVR chain graph that has exactly the minimum set of bidirected edges for its equivalence class. A similar approach is used for the PC-like algorithm. We then compare the results to access the performance of the decomposition-based algorithm against the PC-like algorithm. The entire plots of the error measures and running times can be seen in the supplementary document (Javidian and Valtorta, 2019b). From the plots, we infer that: (a) both algorithms yield better results on sparse graphs \((N = 2, 3)\) than on dense graphs \((N = 5, 8, 10)\), for example see Figures 3.22 and 3.23; (b) for both algorithms, typically the TPR and ACC increase with sample size, for example see Figure 3.22; (c) for both algorithms, typically the SHD decreases with sample size for sparse graphs \((N = 2, 3)\). For \(N = 5\) the SHD decreases with sample size for the decomposition-based algorithm while the SHD has no clear dependence on the sample size for the PC-like algo-
rithm in this case. Typically, for the PC-like algorithm the SHD increases with sample size for dense graphs \((N = 8, 10)\) while the SHD has no clear dependence on the sample size for the decomposition-based algorithm in these cases, for example see Figure 3.23; (d) a large significance level \((\alpha = 0.05)\) typically yields large TPR, FPR, and SHD, for example see Figures 3.22 and 3.23; (e) in almost all cases, the performance of the decomposition-based algorithm based on all error measures i.e., TPR, FPR, ACC, and SHD is better than the performance of the PC-like algorithm, for example see Figure 3.22 and 3.23; (f) In most cases, error measures based on \(\alpha = 0.01\) and \(\alpha = 0.005\) are very close, for example see Figure 3.22 and 3.23. Generally, our empirical results suggests that in order to obtain a better performance, we can choose a small value (say \(\alpha = 0.005\) or 0.01) for the significance level of individual tests along with large sample (say \(n = 3000\) or 10000). However, the optimal value for a desired overall error rate may depend on the sample size, significance level, and the sparsity of the underlying graph.

Considering average running times vs. sample sizes, it can be seen that, for example see Figure 3.24: (a) the average run time increases with sample size; (b) the average run times based on \(\alpha = 0.01\) and \(\alpha = 0.005\) are very close and in all cases are better than \(\alpha = 0.05\), while choosing \(\alpha = 0.005\) yields a consistently (albeit slightly) lower average run time across all the settings in the current simulation; (c) generally, the average run time for the PC-like algorithm is better than that for the decomposition-based algorithm. One possible justification is related to the details of the implementation. The PC algorithm implementation in the pcalg R package is very well optimized, while we have not concentrated on optimizing our implementation of the LCD algorithm; therefore the comparison on run time may be unfair to the new algorithm. For future work, one may consider both optimization of the LCD implementation and instrumentation of the code to allow counting characteristic operations and therefore reducing the dependence of run-time comparison on program optimization. The simulations were run on an Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz. An R language package that implements our algorithms is available in
Figure 3.22: Error measures of the decomposition-based and PC-like algorithms for randomly generated Gaussian chain graph models: average over 25 repetitions with 30 variables. The four rows correspond to N = 2 and 8. The three columns give three error measures: TPR, FPR and ACC in each setting respectively. In each plot, the solid (blue)/dashed (green)/dotted (red) lines correspond to significance levels $\alpha = 0.05/0.01/0.005$. 

112
Figure 3.23: Error measure SHD of the decomposition-based and PC-like algorithms for randomly generated Gaussian chain graph models: average over 25 repetitions with 30 variables. The first row correspond to N = 2, the second row correspond to N = 5, and the third row correspond to N = 8. The first two columns correspond to the essential recovery while the last two columns correspond to the minimum bidirected recovery respectively. In each plot, the solid (blue)/dashed (green)/dotted (red) lines correspond to significance levels $\alpha = 0.05/0.01/0.005$. 

113
Figure 3.24: Running times of the decomposition-based and PC-like algorithms for randomly generated Gaussian chain graph models: average over 25 repetitions with 30 variables correspond to $N = 2$. The first two columns correspond to the essential recovery algorithm while the last two columns correspond to the minimum bidirected recovery respectively. In each plot, the solid (blue)/dashed (green)/dotted (red) lines correspond to significance levels $\alpha = 0.05/0.01/0.005$.

It is worth noting that since our implementation of the decomposition-based algorithms is based on the LCD R package, the generated normal random samples from a given MVR chain graph is not necessarily faithful to it. So, one can expect a better performance if we only consider faithful probability distributions in the experiments. Also, the LCD R package uses $\chi^2$ test which is an asymptotic test for $G^2$ (Ma, Xie, and Geng, 2008). Again, one can expect a better results if we replace the asymptotic test used in the LCD R package with an exact test. However, there is a trade-off between accuracy and computational time (Ma, Xie, and Geng, 2008).
Performance on Discrete Bayesian Networks

Bayesian networks are special cases of MVR chain graphs. It is of interest to see whether the decomposition-based algorithms still work well when the data are actually generated from a Bayesian network. For this purpose, in this subsection, we perform simulation studies for four well-known Bayesian networks from Bayesian Network Repository:

- **ASIA** (Lauritzen and Spiegelhalter, 1988): with 8 nodes, 8 edges, and 18 parameters, it describes the diagnosis of a patient at a chest clinic who may have just come back from a trip to Asia and may be showing dyspnea. Standard learning algorithms are not able to recover the true structure of the network because of the presence of a functional node (either, representing logical or)\(^8\).

Table 3.4: Results for discrete samples from the ASIA network. Each row corresponds to the significance level: \(\alpha = 0.05/0.01/0.005\) respectively.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>TPR</th>
<th>FPR</th>
<th>ACC</th>
<th>SHD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decomposition-Based essential recovery algorithm</td>
<td>0.625</td>
<td>0.2</td>
<td>0.75</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>0.625</td>
<td>0.2</td>
<td>0.75</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>0.625</td>
<td>0.2</td>
<td>0.75</td>
<td>9</td>
</tr>
<tr>
<td>PC-Like essential recovery algorithm Algorithm</td>
<td>0.625</td>
<td>0</td>
<td>0.893</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>0.625</td>
<td>0</td>
<td>0.893</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>0.625</td>
<td>0</td>
<td>0.893</td>
<td>6</td>
</tr>
<tr>
<td>Decomposition-Based Algorithm with Minimum bidirected Edges</td>
<td>0.625</td>
<td>0.2</td>
<td>0.75</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>0.625</td>
<td>0.2</td>
<td>0.75</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>0.625</td>
<td>0.2</td>
<td>0.75</td>
<td>8</td>
</tr>
<tr>
<td>PC-Like Algorithm with Minimum bidirected Edges</td>
<td>0.625</td>
<td>0</td>
<td>0.893</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>0.625</td>
<td>0</td>
<td>0.893</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>0.625</td>
<td>0</td>
<td>0.893</td>
<td>4</td>
</tr>
</tbody>
</table>

- **INSURANCE** (Binder et al., 1997): with 27 nodes, 52 edges, and 984 parameters, it evaluates car insurance risks.

\(^8\)Package 'bnlearn'
Table 3.5: Results for discrete samples from the INSURANCE network. Each row corresponds to the significance level: $\alpha = 0.05/0.01/0.005$ respectively.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>TPR</th>
<th>FPR</th>
<th>ACC</th>
<th>SHD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decomposition-Based essential recovery algorithm</td>
<td>0.635</td>
<td>0.0167</td>
<td>0.932</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td>0.635</td>
<td>0.020</td>
<td>0.926</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>0.654</td>
<td>0.0134</td>
<td>0.937</td>
<td>28</td>
</tr>
<tr>
<td>PC-Like essential recovery algorithm</td>
<td>0.558</td>
<td>0</td>
<td>0.934</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>0.519</td>
<td>0</td>
<td>0.929</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>0.519</td>
<td>0</td>
<td>0.929</td>
<td>37</td>
</tr>
<tr>
<td>Decomposition-Based Algorithm with Minimum bidirected Edges</td>
<td>0.635</td>
<td>0.0167</td>
<td>0.932</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>0.635</td>
<td>0.020</td>
<td>0.926</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>0.654</td>
<td>0.0134</td>
<td>0.937</td>
<td>27</td>
</tr>
<tr>
<td>PC-Like Algorithm with Minimum bidirected Edges</td>
<td>0.558</td>
<td>0</td>
<td>0.934</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>0.519</td>
<td>0</td>
<td>0.929</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>0.519</td>
<td>0</td>
<td>0.929</td>
<td>27</td>
</tr>
</tbody>
</table>

- ALARM (Beinlich et al., 1989): with 37 nodes, 46 edges and 509 parameters, it was designed by medical experts to provide an alarm message system for intensive care unit patients based on the output a number of vital signs monitoring devices.

Table 3.6: Results for discrete samples from the ALARM network. Each row corresponds to the significance level: $\alpha = 0.05/0.01/0.005$ respectively.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>TPR</th>
<th>FPR</th>
<th>ACC</th>
<th>SHD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decomposition-Based essential recovery algorithm</td>
<td>0.783</td>
<td>0.0194</td>
<td>0.967</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>0.783</td>
<td>0.0161</td>
<td>0.967</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>0.761</td>
<td>0.021</td>
<td>0.964</td>
<td>36</td>
</tr>
<tr>
<td>PC-Like essential recovery algorithm</td>
<td>0.457</td>
<td>0</td>
<td>0.962</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>0.435</td>
<td>0</td>
<td>0.961</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>0.413</td>
<td>0</td>
<td>0.959</td>
<td>41</td>
</tr>
<tr>
<td>Decomposition-Based Algorithm with Minimum bidirected Edges</td>
<td>0.783</td>
<td>0.0194</td>
<td>0.967</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>0.783</td>
<td>0.0161</td>
<td>0.967</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>0.761</td>
<td>0.021</td>
<td>0.964</td>
<td>35</td>
</tr>
<tr>
<td>PC-Like Algorithm with Minimum bidirected Edges</td>
<td>0.457</td>
<td>0</td>
<td>0.962</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>0.435</td>
<td>0</td>
<td>0.961</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>0.413</td>
<td>0</td>
<td>0.959</td>
<td>36</td>
</tr>
</tbody>
</table>

- HAILFINDER (Abramson et al., 1996): with 56 nodes, 66 edges, and 2656 parame-
ters, it was designed to forecast severe summer hail in northeastern Colorado.

Table 3.7: Results for discrete samples from the HAILFINDER network. Each row corresponds to the significance level: $\alpha = 0.05/0.01/0.005$ respectively.

<table>
<thead>
<tr>
<th></th>
<th>TPR</th>
<th>FPR</th>
<th>ACC</th>
<th>SHD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decomposition-Based essential recovery algorithm</td>
<td>0.758</td>
<td>0.003</td>
<td>0.986</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>0.742</td>
<td>0.002</td>
<td>0.987</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>0.757</td>
<td>0.002</td>
<td>0.988</td>
<td>22</td>
</tr>
<tr>
<td>PC-Like essential recovery algorithm</td>
<td>0.457</td>
<td>0</td>
<td>0.962</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>0.515</td>
<td>0.0007</td>
<td>0.979</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>0.515</td>
<td>0.0007</td>
<td>0.979</td>
<td>40</td>
</tr>
<tr>
<td>Decomposition-Based Algorithm with Minimum bidirected Edges</td>
<td>0.758</td>
<td>0.003</td>
<td>0.986</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>0.742</td>
<td>0.002</td>
<td>0.987</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>0.757</td>
<td>0.002</td>
<td>0.988</td>
<td>24</td>
</tr>
<tr>
<td>PC-Like Algorithm with Minimum bidirected Edges</td>
<td>0.457</td>
<td>0</td>
<td>0.962</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>0.515</td>
<td>0.0007</td>
<td>0.979</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>0.515</td>
<td>0.0007</td>
<td>0.979</td>
<td>39</td>
</tr>
</tbody>
</table>

We compare the performance of our algorithms against the PC-like algorithm for these Bayesian networks for three different significance levels ($\alpha = 0.05/0.01/0.005$).

The results of all learning methods are summarized in Table 3.4, 3.5, 3.6, and 3.7. For the decomposition-based methods, all the three error measures: TPR, FPR and SHD are similar to those of the PC-like algorithms, but the results indicate that the decomposition-based method outperforms the PC-like algorithms as the size of Bayesian network become larger, especially in terms of TPR and SHD.

3.6 Discussion and Conclusion

In this paper, we presented a computationally feasible algorithm for learning the structure of MVR chain graphs via decomposition. We compared the performance of our algorithm with that of the PC-like algorithm proposed by (Sonntag and Peña, 2012), in the Gaussian and discrete cases. The PC-like algorithm is a constraint-based algorithm that learns the structure of the underlying MVR chain graph in four steps: (a) determining the skeleton:
the resulting undirected graph in this phase contains an undirected edge \( u - v \) iff there is no set \( S \subseteq V \setminus \{ u, v \} \) such that \( u \perp \! \! \perp v \mid S \); (b) determining the v-structures (unshielded colliders); (c) orienting some of the undirected/directed edges into directed/bidirected edges according to a set of rules applied iteratively; (d) transforming the resulting graph in the previous step into an MVR CG. The essential recovery algorithm obtained after step (c) contains all directed and bidirected edges that are present in every MVR CG of the same Markov equivalence class. The decomposition-based algorithm is also a constraint-based algorithm that is based on a divide and conquer approach and contains four steps: (a) determining the skeleton by a divide-and-conquer approach; (b) determining the v-structures (unshielded colliders) with localized search for \( m \)-separators; continuing with steps (c) and (d) exactly as in the PC-like algorithm. The correctness of both algorithms lies upon the assumption that the probability distribution \( p \) is faithful to some MVR CG. As for the PC-like algorithms, unless the probability distribution \( p \) of the data is faithful to some MVR CG the learned CG cannot be ensured to factorize \( p \) properly. Empirical simulations in the Gaussian case show that both algorithms yield good results when the underlying graph is sparse. The decomposition-based algorithm achieves competitive results with the PC-like learning algorithm in both Gaussian and discrete cases. In fact, the decomposition-based method usually outperforms the PC-like algorithm in all four error measures i.e., TPR, FPR, ACC, and SHD. Such simulation results confirm that our method is reliable both when latent variables are present (and the underlying graph is an MVR CG) and when there are no such variables (and the underlying graph is a DAG. The algorithm works reliably when latent variables are present and only fails when selection bias variables are presents. Informally, our algorithm allows relaxing the causal sufficiency assumption, because only selection bias needs to be represented explicitly. Since our implementation of the decomposition-based algorithm is based on the LCD R package, with fixed number of samples, one can expect a better performance if we replace the asymptotic test used in the LCD R package with an exact test. However, there is a trade-off between accuracy and computational time.
Also, one can expect a better results if we only consider faithful probability distributions in the experiments.

The natural continuation of the work presented here would be to develop a learning algorithm with weaker assumptions than the one presented. This could for example be a learning algorithm that only assumes that the probability distribution satisfies the composition property. It should be mentioned that (Peña, Sonntag, and Nielsen, 2014) developed an algorithm for learning LWF CGs under the composition property. However, (Peña, 2014a) proved that the same technique cannot be used for MVR chain graphs.
CHAPTER 4

AMP CHAIN GRAPHS

This chapter deals with chain graphs under the alternative Andersson-Madigan-Perlman (AMP) interpretation (Andersson, Madigan, and Perlman, 1996, 2001). AMP CGs are useful when we have a set of variables for which the internal relations has no causal ordering, so the relations should be modelled as a Markov network, but also a second set of variables which can be seen as causes for some of these variables in the first set. The internal structure of the first set of variables can then be modelled as a Markov network, creating a chain component in an AMP CG, and the causes as parents of some of the variables in the chain component. Note that for AMP CGs the parents only affects the direct children in the chain component, not all the nodes in the chain component as in the case of LWF CGs. An example in medicine (Sonntag and Peña, 2015b) when such a model might be appropriate is when we are modelling pain levels on different areas on the body of a patient. The pain levels can then be seen as correlated "geographically" over the body, and hence be modelled as a Markov network. Certain other factors do, however, exist that alters the pain levels locally at some of these areas, such as the type of body part the area is located on or if local anaesthetic has been administered in that area and so on. These outside factors can then be modelled as parents affecting the pain levels locally. AMP chain graphs widely studied in in different areas from applications in biology (Sonntag and Peña, 2015b), to more advanced theoretical investigations (Richardson, 1998; Levitz, Perlman, and Madigan, 2001; Roverato, 2005; Roverato and Rocca, 2006; Drton, 2009; Studený, Roverato, and Š. Štěpánová, 2009; Peña, 2014b, 2015; Sonntag and Peña, 2015b; Peña, 2016b; Peña and Gómez-Olmedo, 2016; Peña, 2018b,a).
In this chapter, we address the problem of finding a minimal separator in an Andersson-Madigan-Perlman chain graph (AMP CG), namely, finding a set $Z$ of nodes that separates a given non-adjacent pair of nodes such that no proper subset of $Z$ separates that pair. We analyze several versions of this problem and offer polynomial time algorithms for each. These include finding a minimal separator from a restricted set of nodes, finding a minimal separator for two given disjoint sets, and testing whether a given separator is minimal. We apply these results to extend the decomposition approach for learning Bayesian networks (BNs) proposed by (Xie, Zheng, and Zhao, 2006) to learn AMP CGs, which include BNs as a special case, under the faithfulness assumption. The advantages of this decomposition approach hold in the more general setting: reduced complexity and increased power of computational independence tests. We show that the PC-like algorithm is order-dependent, in the sense that the output can depend on the order in which the variables are given. We propose two modifications of the PC-like algorithm that remove part or all of this order-dependence. Simulations under a variety of settings demonstrate the competitive performance of our decomposition based method in comparison with the (modified version of) PC-like algorithm. In fact, the decomposition-based algorithm usually outperforms the PC-like algorithm. We empirically show that both algorithms work very well when the underlying graph is sparse.

4.1 Basic Definitions and Concepts

In this section, we consider graphs containing both directed ($\rightarrow$) and undirected ($-$) edges and largely use the terminology of (Andersson, Madigan, and Perlman, 2001), where the reader can also find further details. Below we briefly list some of the central concepts used in this chapter.

If $A \subseteq V$ is a subset of the vertex set in a graph $G = (V, E)$, the *induced subgraph* $G_A = (A, E_A)$ is a graph in which the edge set $E_A = E \cap (A \times A)$ is obtained from $G$ by keeping edges with both endpoints in $A$. 

121
If there is an arrow from $a$ pointing towards $b$, $a$ is said to be a parent of $b$. The set of parents of $b$ is denoted as $pa(b)$. If there is an undirected edge between $a$ and $b$, $a$ and $b$ are said to be adjacent or neighbors. The set of neighbors of a vertex $a$ is denoted as $ne(a)$. The expressions $pa(A)$ and $ne(A)$ denote the collection of parents and neighbors of vertices in $A$ that are not themselves elements of $A$. The boundary $bd(A)$ of a subset $A$ of vertices is the set of vertices in $V \setminus A$ that are parents or neighbors to vertices in $A$. The closure of $A$ is $cl(A) = bd(A) \cup A$.

A path of length $n$ from $a$ to $b$ is a sequence $a = a_0, \ldots, a_n = b$ of distinct vertices such that $(a_i, a_{i+1}) \in E$, for all $i = 1, \ldots, n$. A chain of length $n$ from $a$ to $b$ is a sequence $a = a_0, \ldots, a_n = b$ of distinct vertices such that $(a_i, a_{i+1}) \in E$, or $(a_{i+1}, a_i) \in E$, or $\{a_i, a_{i+1}\} \in E$, for all $i = 1, \ldots, n$. A vertex $a$ is said to be an ancestor of a vertex $b$ if either there is a directed path $a \to \cdots \to b$ from $a$ to $b$. We define the smallest ancestral set containing $A$ as $An(A) := \{a : a$ is an ancestor of $b$ for some $b \in A\}$. A vertex $a$ is said to be anterior to a vertex $b$ if there is a chain $\mu$ from $a$ to $b$ on which every edge is either of the form $\gamma \rightarrow \delta$, or $\gamma \leftarrow \delta$ with $\delta$ between $\gamma$ and $b$, or $a = b$; that is, there are no edges $\gamma \leftarrow \delta$ pointing toward $a$. We apply this definition to sets: $ant(X) = \{a : a$ is an anterior of $b$ for some $b \in X\}$.

A partially directed cycle (or semi-directed cycle) in a graph $G$ is a sequence of $n$ distinct vertices $v_1, v_2, \ldots, v_n (n \geq 3)$, and $v_{n+1} \equiv v_1$, such that

(a) for all $i (1 \leq i \leq n)$ either $v_i - v_{i+1}$ or $v_i \rightarrow v_{i+1}$, and

(b) there exists a $j (1 \leq j \leq n)$ such that $v_j \rightarrow v_{j+1}$.

An AMP chain graph is a graph in which there are no partially directed cycles. The chain components $T$ of a chain graph are the connected components of the undirected graph obtained by removing all directed edges from the chain graph. We define the smallest coherent set containing $A$ as $Co(A) := \bigcup \{\tau : \tau \in T \mid \tau \cap A \neq \emptyset\}$. Let $\overline{G}$ be obtained by deleting all directed edges of $G$; for $A \subseteq V$ the extended subgraph $G[A]$ is defined by $G[A] := G_{An(A)} \cup \overline{G}_{Co(An(A))}$.

A triple of vertices $\{X, Y, Z\}$ is said to form a triplex in $CG$ if the induced subgraph
Figure 4.1: (a) Triplexes and (b) the corresponding augmented triplex, (c) the four configurations that define the bi-flag; (d) the corresponding augmented bi-flag. The “?” indicates that either \( X \rightarrow Y \in G \), \( X \rightarrow Y \in G \), \( Y \rightarrow X \in G \), or \( X \) and \( Y \) are not adjacent in \( G \).

\( CG_{X \cup Y \cup Z} \) is either \( X \rightarrow Y \rightarrow Z \), \( X \rightarrow Y \leftarrow Z \), or \( X \rightarrow Y \leftarrow Z \). A triplex is augmented by adding the \( X \rightarrow Z \) edge. A set of four vertices \( \{X, A, B, Y\} \) is said to form a bi-flag if the edges \( X \rightarrow A \), \( Y \rightarrow B \), and \( A \leftarrow B \) are present in the induced subgraph over \( \{X, A, B, Y\} \). A bi-flag is augmented by adding the edge \( X \leftarrow Y \). A minimal complex (or simply a complex) in a chain graph is an induced subgraph of the form \( a \rightarrow v_1 \leftarrow \cdots \leftarrow v_r \leftarrow b \). The augmented \( CG G^* \) is the undirected graph formed by augmenting all triplexes and bi-flags in \( CG \) and replacing all directed edges with undirected edges (see Fig. 4.1). The skeleton (underlying graph) of a \( CG G \) is obtained from \( G \) by changing all directed edges of \( G \) into undirected edges. Vertex \( Y \) is an unshielded collider (or V-structure) in a DAG \( G \) if \( G \) contains the induced subgraph \( U \rightarrow Y \leftarrow V \).

Given an undirected graph \( G = (V, E) \), a subset \( S \subseteq V \) that does not contain \( a \) or \( b \) is said to be an \((a, b)\)-separator if all paths from \( a \) to \( b \) intersect \( S \). A set \( S \) of nodes that separates a given pair of nodes such that no proper subset of \( S \) separates that pair is called a minimal separator. Note that removing an \((a, b)\)-separator disconnects a graph into two connected components, one containing \( a \), and another containing \( b \). Conversely, if a
set $S$ disconnects a graph into a connected component including $a$ and another connected component including $b$, then $S$ is an $(a, b)$-separator. Two disjoint vertex subsets $A$ and $B$ of $V$ are adjacent if there is at least one pair of adjacent vertices $u \in A$ and $v \in B$. Let $A$ and $B$ be two disjoint non-adjacent subsets of $V$. Similarly, we define an $(A, B)$-separator to be any subset of $V \setminus (A \cup B)$ whose removal separates $A$ and $B$ in distinct connected components. A minimal $(A, B)$-separator does not contain any other $(A, B)$-separator.

4.2 On the Properties of AMP Chain Graphs

In this section an alternative Markov property (AMP) for CGs is introduced that is a natural extension of the moralization criterion, called augmentation, to AMP chain graphs. We then introduce an extension of Pearl’s $d$-separation criterion, called $p$-separation, which is equivalent to the global property resulting from the augmentation for arbitrary distributions.

**Definition 4.1.** (Global Markov property for AMP chain graphs) For any triple $(A, B, S)$ of disjoint subsets of $V$ such that $S$ separates $A$ from $B$ in $(G[A \cup B \cup S])^a$, in the augmented graph of the extended subgraph of $A \cup B \cup S$, we have $A \perp \perp B | S$ (or $\langle A, B | S \rangle$) i.e., $A$ is independent of $B$ given $S$.

An equivalent pathwise separation criterion that identifies all valid conditional independencies under the AMP Markov property was introduced in (Levitz, Perlman, and Madigan, 2001):

**Definition 4.2.** (The pathwise $p$-separation criterion for AMP chain graphs) A node $B$ in a chain $\rho$ in an AMP CG $G$ is called a triplex node in $\rho$ if $A \rightarrow B \leftarrow C$, $A \rightarrow B \rightarrow C$, or $A - B \leftarrow C$ is a subchain of $\rho$. Moreover, $\rho$ is said to be $Z$-open with $Z \subseteq V$ when

- every triplex node in $\rho$ is in $An(Z)$, and
- every non-triplex node $B$ in $\rho$ is outside $Z$, unless $A - B - C$ is a subchain of $\rho$ and $pa_G(B) \setminus Z \neq \emptyset$. 

124
Let $X, Y \neq \emptyset$ and $Z$ (may be empty) denote three disjoint subsets of $V$. When there is no $Z$-open chain in an AMP CG $G$ between a node in $X$ and a node in $Y$, we say that $X$ is separated from $Y$ given $Z$ in $G$ and denote it as $X \perp \perp Y | Z$.

Theorem 4.1 in (Levitz, Perlman, and Madigan, 2001) establishes the equivalence of the $p$-separation criterion and the augmentation criterion occurring in the AMP global Markov property for CGs.

**Example 15.** Consider the AMP CG $G$ in Fig. 4.2(a). The global Markov property of AMP chain graphs implies that $X \perp \perp Y | A$ (see Fig. 4.2). There is no $A$-open chain in the AMP CG $G$ between $X$ and $Y$ because the only chain between $X$ and $Y$ i.e., $X \rightarrow A \leftarrow B \leftarrow Y$ is blocked at $B$ ($B$ is a triplex node in the chain and $B \notin An(A)$).

We say that two AMP CGs $G$ and $H$ are *Markov equivalent* or that they are in the same *Markov equivalence class* if they induce the same conditional independence restrictions. Two chain graphs $G$ and $H$ are Markov equivalent if and only if they have the same skeletons and the same triplexes (Andersson, Madigan, and Perlman, 2001). Two LWF chain
graphs $G$ and $H$ are Markov equivalent if and only if they have the same skeletons and the same minimal complexes (Frydenberg, 1990). Two DAGs $G$ and $H$ are Markov equivalent if and only if they have the same skeletons and the same unshielded colliders (Pearl, 1988). The condition for AMP Markov equivalence of CGs more closely resembles that for DAG Markov equivalence than does the condition for LWF Markov equivalence of CGs, in the sense that triplexes involve only three vertices, while complexes can involve arbitrarily many vertices.

We say that AMP chain graphs $G$ and $H$ belong to the same strong Markov equivalent class iff $G$ and $H$ are Markov equivalent and contain the same flags. An AMP CG $G^*$ is said to be the AMP essential graph of its Markov equivalence class iff for every directed edge $A \to B$ that exists in $G^*$ there exists no AMP CG $H$ s.t. $G^*$ and $H$ are Markov equivalent and $A \leftarrow B$ is in $H$. An AMP CG $G^*$ is said to be the largest deflagged graph of its Markov equivalence class iff there exists no other AMP CG $H$ s.t. $G^*$ and $H$ are Markov equivalent and either $H$ contains fewer flags than $G^*$ or $G^*$ and $H$ belong to the same strong Markov equivalence class but $H$ contains more undirected edges. Any largest deflagged graph or AMP essential graph are AMP CGs and both of these have been proven to be unique for the Markov equivalence class they represent (Roverato and Rocca, 2006; Andersson and Perlman, 2006).

### 4.3 Finding Minimal Separators in AMP Chain Graphs

In this section we propose and solve an optimization problem related to the separation in AMP chain graphs. The basic problem may be formulated as follows: given a pair of non-adjacent nodes, $x$ and $y$, in an AMP chain graph, $G$, find a minimal set of nodes that separates $x$ and $y$. We analyze several versions of this problem and offer polynomial time algorithms for each. These include the following problems:

**Problem 13.** (test for minimal separation) Given two non-adjacent nodes $X$ and $Y$ in an AMP chain graph $G$ and a set $Z$ that separates $X$ from $Y$, test if $Z$ is minimal i.e., no proper
subset of $Z$ separates $X$ from $Y$.

**Problem 14.** (minimal separation) Given two non-adjacent nodes $X$ and $Y$ in an AMP chain graph $G$, find a minimal separating set between $X$ and $Y$, namely, find a set $Z$ such that $Z$, and no proper subset of $Z$, separates $X$ from $Y$.

**Problem 15.** (restricted separation) Given two non-adjacent nodes $X$ and $Y$ in an AMP chain graph $G$ and a set $S$ of nodes not containing $X$ and $Y$, find a subset $Z$ of $S$ that separates $X$ from $Y$.

**Problem 16.** (restricted minimal separation) Given two non-adjacent nodes $X$ and $Y$ in an AMP chain graph $G$ and a set $S$ of nodes not containing $X$ and $Y$, find a subset $Z$ of $S$ which is minimal and separates $X$ from $Y$.

**Problem 17.** (minimal separation of two disjoint non-adjacent sets) Given two disjoint non-adjacent sets $X$ and $Y$ in an AMP chain graph $G$, find a minimal separating set between $X$ and $Y$, namely, find a set $Z$ such that $Z$, and no proper subset of $Z$, separates $X$ from $Y$.

**Problem 18.** (enumeration of all minimal separators) Given two non-adjacent nodes (or disjoint subsets) $X$ and $Y$ in an AMP chain graph $G$, enumerate all minimal separating sets between $X$ and $Y$.

We prove that it is possible to transform our problem into a separation problem, where the undirected graph in which we have to look for the minimal set separating $X$ from $Y$ depends only on $X$ and $Y$. For each above mentioned problem, we propose and analyze an algorithm that, taking into account the previous results, solves it.

4.3.1 Main Theorem

In this subsection we prove that it is possible to transform our problem into a separation problem, where the undirected graph in which we have to look for the minimal set separat-
ing $X$ from $Y$ depends only on $X$ and $Y$. Later, in the next subsections, we will apply this result to developing an efficient algorithm that solves our problems.

The next proposition shows that if we want to test a separation relationship between two disjoint sets of nodes $X$ and $Y$ in an AMP chain graph, where the separating set is included in the anterior set of $X \cup Y$, then we can test this relationship in a smaller AMP chain graph, whose set of nodes is formed only by the anteriors of $X$ and $Y$.

**Proposition 4.3.** Given an AMP chain graph $G = (V, E)$. Consider that $X$, $Y$, and $Z$ are three disjoint subsets of $V$, $Z \subseteq \text{ant}(X \cup Y)$, and $H = G_{\text{ant}(X \cup Y)}$ is the subgraph of $G$ induced by $\text{ant}(X \cup Y)$. Then $\langle X, Y | Z \rangle_G \iff \langle X, Y | Z \rangle_H$.

**Proof.** ($\Rightarrow$) The necessary condition is obvious, because a separator in a graph is also a separator in all of its subgraphs.

($\Leftarrow$) Since $bd(\text{ant}(X \cup Y)) = \emptyset$, so $Co(An(\text{ant}(X \cup Y))) = \text{ant}(X \cup Y)$. Let $\langle X, Y | Z \rangle_H$ and $Z \subseteq \text{ant}(X \cup Y)$, then $Co(An(X \cup Y \cup Z)) \subseteq \text{ant}(X \cup Y)$. Consider that $\langle X, Y | Z \rangle_G$. This means that $X$ is not separated from $Y$ given $Z$ in $(G[X \cup Y \cup Z])^a$, which is a subgraph of $(G[\text{ant}(X \cup Y)])^a$. In other words, there is a chain $C$ between $X$ and $Y$ in $H^a = (G[\text{ant}(X \cup Y)])^a$ that bypasses $Z$. Once again using $Z \subseteq \text{ant}(X \cup Y)$, we obtain that $X$ and $Y$ are not separated by $Z$ in $H$, in contradiction to the assumption $\langle X, Y | Z \rangle_H$. Therefore, it has to be $\langle X, Y | Z \rangle_G$. □

The following proposition establishes the basic result necessary to solve our optimization problems.

**Proposition 4.4.** Given an AMP chain graph $G = (V, E)$. Consider that $X$, $Y$, and $Z$ are three disjoint subsets of $V$ such that $\langle X, Y | Z \rangle$ and $\langle X, Y | Z' \rangle$, $\forall Z' \subseteq Z$. Then $Z \subseteq \text{ant}(X \cup Y)$.

**Proof.** Suppose that $Z \notin \text{ant}(X \cup Y)$. Define $Z' = Z \cap \text{ant}(X \cup Y)$. Then, by assumption we have $\langle X, Y | Z' \rangle$. Since $Z' \subseteq \text{ant}(X \cup Y)$, it is obvious that $Co(An(X \cup Y \cup Z')) \subseteq \text{ant}(X \cup Y)$. So, $X$ and $Y$ are not separated by $Z'$ in $(G[X \cup Y \cup Z'])^a$, hence there is a chain $C$ between $X$
and \( Y \) in \((G[X ∪ Y ∪ Z'])^a\) that bypasses \( Z \) i.e., the chain \( C \) is formed from nodes in \( \text{ant}(X ∪ Y) \) that are outside of \( Z \). Since \( \text{Co}(\text{An}(X ∪ Y ∪ Z')) \subseteq \text{ant}(X ∪ Y) \), then \((G[X ∪ Y ∪ Z'])^a\) is a subgraph of \((G[\text{ant}(X ∪ Y)])^a\). Then, the previously found chain \( C \) is also a chain in \((G[\text{ant}(X ∪ Y)])^a\) that bypasses \( Z \), which means that \( X \) and \( Y \) are not separated by \( Z \) in \((G[\text{ant}(X ∪ Y)])^a = (G[\text{ant}(X∪Y)])^a\). So, \( X \) and \( Y \) are not \( p \)-separated by \( Z \) in \( G_{\text{ant}(X∪Y)} \). This implies that \( X \) and \( Y \) are not \( p \)-separated by \( Z \) in \( G \), in contradiction to the assumption \( \langle X, Y | Z \rangle \). Therefore, it has to be \( Z \subseteq \text{ant}(X ∪ Y) \). □

The next proposition shows that, by combining the results in propositions 4.3 and 4.4, we can reduce our problems to a simpler one, which involves a smaller graph.

**Proposition 4.5.** Let \( G = (V, E) \) be an AMP chain graph, and \( X, Y \subseteq V \) are two disjoint subsets. Then the problem of finding a minimal separating set for \( X \) and \( Y \) in \( G \) is equivalent to the problem of finding a minimal separating set for \( X \) and \( Y \) in the induced subgraph \( G_{\text{ant}(X∪Y)} \).

**Proof.** The proof is very similar to the proof of Proposition 3 in (Acid and Campos, 1996; Javidian and Valtorta, 2018a) and Proposition 9 in (Javidian and Valtorta, 2018b). Let \( H = G_{\text{ant}(X∪Y)} \), and let us to define sets \( S_G = \{Z \subseteq V | \langle X, Y | Z \rangle_G \} \) and \( S_H = \{Z \subseteq \text{ant}(X ∪ Y) | \langle X, Y | Z \rangle_H \} \). Then we have to prove that \( \min_{Z \in S_G} |Z| = \min_{Z \in S_H} |Z| \), and therefore, by proposition 4.4, the sets of minimal separators are the same. From proposition 4.3, we deduce that \( S_H \subseteq S_G \), and therefore \( \min_{Z \in S_H} |Z| \geq \min_{Z \in S_G} |Z| \).

(⇒) Let \( T = \min(Z \in S_G) \). Then \( \forall T' \subseteq T \) we have \( T' \notin S_G \), and from proposition 4.4 we obtain \( T \subseteq \text{ant}(X ∪ Y) \), and now using proposition 4.3 we get \( T \in S_H \). So, we have \( |T| = \min_{Z \in S_H} |Z| \geq \min_{Z \in S_G} |Z| = |T| \), hence \( |T| = \min_{Z \in S_H} |Z| \).

(⇐) Let \( T = \min(Z \in S_H) \). If \( |T| = \min_{Z \in S_H} |Z| > \min_{Z \in S_G} |Z| = |Z_0| \), we have \( \forall Z' \subseteq Z_0, Z' \notin S_G \), and therefore, once again using proposition 4.4 and 4.3, we get \( Z_0 \in S_H \), so that \( |Z_0| \geq \min_{Z \in S_H} |Z| = |T| \), which is a contradiction. Thus, \( |T| = \min_{Z \in S_G} |Z| \). □
**Theorem 4.6.** The problem of finding a minimal separating set for X and Y in an AMP chain graph $G$ is equivalent to the problem of finding a minimal separating set for X and Y in the undirected graph $(G_{ant(X\cup Y)})^a$.

**Proof.** The proof is very similar to the proof of Theorem 1 in (Acid and Campos, 1996; Javidian and Valtorta, 2018a) and Theorem 10 in (Javidian and Valtorta, 2018b). Using the same notation from proposition 4.5, let $H^a$ be the augmented graph of $H = G_{ant(X\cup Y)}$, and $S^a_H = \{ Z \subseteq ant(X \cup Y)(X, Y|Z)_{H^a}\}$. Let $Z$ be any subset of ant$(X \cup Y)$. Then taking into account the characteristics of anterior sets, it is clear that $H_{ant(X\cup Y\cup Z)} = H$. Then, we have $Z \in S_H \Leftrightarrow (X, Y|Z)_H \Leftrightarrow (X, Y|Z)_{(H_{ant(X\cup Y\cup Z)})^a} \Leftrightarrow (X, Y|Z)_{H^a} \Leftrightarrow Z \in S^a_H$. Hence, $S_H = S^a_H$. □

![Figure 4.3: Finding a minimal separator in an AMP chain graph.](image)

Now, using proposition 4.5, we obtain $|T| = \min_{Z \in S_G} |Z| \Leftrightarrow |T| = \min_{Z \in S^a_H} |Z|$. □

### 4.3.2 Algorithms for Finding Minimal Separators

In undirected graphs we have efficient methods of testing whether a separation set is minimal, which are based on the following criterion.

**Theorem 4.7.** Given two nodes $X$ and $Y$ in an undirected graph, a separating set $Z$ between $X$ and $Y$ is minimal if and only if for each node $u$ in $Z$, there is a path from $X$ to $Y$ which passes through $u$ and does not pass through any other nodes in $Z$.

**Proof.** See the proof of Theorem 5 in (Tian, Paz, and Pearl, 1998). □
This theorem leads to Algorithm 14 for Problem 13. The idea is that if $Z$ is minimal then all nodes in $Z$ can be reached using Breadth First Search (BFS) that starts from both $X$ and $Y$ without passing through any other nodes in $Z$.

**Algorithm 14**: Test for minimal separation (Problem 13)

**Input**: A set $Z$ that separates two non-adjacent nodes $X$, $Y$ in the AMP chain graph $G$.

**Output**: If $Z$ is minimal then the algorithm returns TRUE otherwise, returns FALSE.

```
1 if $Z$ contains a node that is not in ant$(X \cup Y)$ then
   2 return FALSE;
else
   4 Construct $G_{ant}(X \cup Y)$;
   5 Construct $(G_{ant}(X \cup Y))^a$;
   6 Starting from $X$, run BFS. Whenever a node in $Z$ is met, mark it if it is not already marked, and do not continue along that path. When BFS stops;
   7 if not all nodes in $Z$ are marked then
      8 return FALSE;
   else
      10 Remove all markings. Starting from $Y$, run BFS. Whenever a node in $Z$ is met, mark it if it is not already marked, and do not continue along that path. When BFS stops;
      11 if not all nodes in $Z$ are marked then
         12 return FALSE;
      else
         14 return TRUE;
   end
end
```

**Analysis of Algorithm 14** (Tian, Paz, and Pearl, 1998): Let $H = G_{ant}(X \cup Y)$ and $|E_H|$ stands for the number of edges in $H^a = (G_{ant}(X \cup Y))^a$. Step 4-5 each requires $O(|E_H^a|)$ time. Thus, the complexity of Algorithm 14 is $O(|E_H|)$.

A variant of Algorithm 14 solves Problem 14.

**Analysis of Algorithm 15**: Each one of steps 2-5 each requires $O(|E_H^a|)$ time. Thus, the overall complexity of Algorithm 15 is $O(|E_H^a|)$.

**Theorem 4.8.** Given two nodes $X$ and $Y$ in an AMP chain graph $G$ and a set $S$ of nodes not containing $X$ and $Y$, there exists some subset of $S$ which separates $X$ and $Y$ if only if
Algorithm 15: Minimal separation (Problem 14)

**Input:** Two non-adjacent nodes \( X, Y \) in the AMP chain graph \( G \).

**Output:** Set \( Z \), that is a minimal separator for \( X, Y \).

1. Construct \( G_{\text{ant}(X \cup Y)} \);
2. Construct \( (G_{\text{ant}(X \cup Y)})^a \);
3. Set \( Z' \) to be \( \text{ne}(X) \) (or \( \text{ne}(Y) \)) in \( (G_{\text{ant}(X \cup Y)})^a \);
   /* \( Z' \) is a separator because, according to the local Markov property of an undirected graph, a vertex is conditionally independent of all other vertices in the graph, given its neighbors (Lauritzen, 1996). */
4. Starting from \( X \), run BFS. Whenever a node in \( Z' \) is met, mark it if it is not already marked, and do not continue along that path. When BFS stops, let \( Z'' \) be the set of nodes which are marked. Remove all markings;
5. Starting from \( Y \), run BFS. Whenever a node in \( Z'' \) is met, mark it if it is not already marked, and do not continue along that path. When BFS stops, let \( Z \) be the set of nodes which are marked;
6. return \( Z \);

the set \( S' = S \cap \text{ant}(X \cup Y) \) separates \( X \) and \( Y \).

**Proof.** (\( \Rightarrow \)) Proof by contradiction. Let \( S' = S \cap \text{ant}(X \cup Y) \) and \( \langle X, Y \, \| S' \rangle \). Since \( S' \subseteq \text{ant}(X \cup Y) \), it is obvious that \( \text{ant}(X \cup Y \cup S') = \text{ant}(X \cup Y) \). So, \( X \) and \( Y \) are not separated by \( S' \) in \( (G_{\text{ant}(X \cup Y)})^a \), hence there is a chain \( C \) between \( X \) and \( Y \) in \( (G_{\text{ant}(X \cup Y)})^a \) that bypasses \( S' \) i.e., the chain \( C \) is formed from nodes in \( \text{ant}(X \cup Y) \) that are outside of \( S \). Since \( \text{ant}(X \cup Y) \subseteq \text{ant}(X \cup Y \cup S'') \), \( \forall S'' \subseteq S \), then \( (G_{\text{ant}(X \cup Y)})^a \) is a subgraph of \( (G_{\text{ant}(X \cup Y \cup S')})^a \). Then, the previously found chain \( C \) is also a chain in \( (G_{\text{ant}(X \cup Y \cup S'')})^a \) that bypasses \( S'' \), which means that \( X \) and \( Y \) are not separated by any \( S'' \subseteq S \) in \( (G_{\text{ant}(X \cup Y \cup S)})^a \), which is a contradiction.

![Figure 4.4: Finding restricted separator in an AMP chain graph.](image-url)

132
Therefore, Problem 15 is solved by testing if $S' = S \cap \text{ant}(X \cup Y)$ separates $X$ and $Y$.

**Algorithm 16: Restricted separation (Problem 15)**

- **Input:** A set $S$ of nodes not containing $X$ and $Y$ in the AMP chain graph $G$.
- **Output:** If there is a subset of $S$ that separates $X$ from $Y$ then the algorithm returns $Z \subseteq S$ that separates $X$ from $Y$ otherwise, returns FALSE.

1. Construct $G_{\text{ant}(X \cup Y)}$;
2. Construct $(G_{\text{ant}(X \cup Y)})^a$;
3. Set $S' = S \cap \text{ant}(X \cup Y)$;
4. Remove $S'$ from $(G_{\text{ant}(X \cup Y)})^a$;
5. Starting from $X$, run BFS;
6. if $Y$ is met then
   7. return FALSE
7. else
   8. return $Z = S'$
10. end

**Analysis of Algorithm 16:** This requires $O(|E_H^a|)$ time.

According to Theorem 4.8, Problem 16 is solved using Algorithm 16 and then, if False not returned, Algorithm 15 with $Z' = S \cap \text{ant}(X \cup Y)$. The time complexity of this algorithm is also $O(|E_H^a|)$.

In order to solve Problem 17, i.e., to find the minimal set separating two disjoint non-adjacent subsets of nodes $X$ and $Y$ (instead of two single nodes) in an AMP chain graph $G$, first we build the undirected graph $(G_{\text{ant}(X \cup Y)})^a$. Next, starting out from this graph, we construct a new undirected graph $\text{Aug}(G : \alpha_X, \alpha_Y)$ by adding two artificial (dummy) nodes $\alpha_X, \alpha_Y$, and connect them to those nodes that are adjacent to some node in $X$ and $Y$, respectively. So, the separation of $X$ and $Y$ in $(G_{\text{ant}(X \cup Y)})^a$ is equivalent to the separation of $\alpha_X$ and $\alpha_Y$ in $\text{Aug}(G : \alpha_X, \alpha_Y)$. Moreover, the minimal separating set for $\alpha_X$ and $\alpha_Y$ in $\text{Aug}(G : \alpha_X, \alpha_Y)$ cannot contain nodes from $(X \cup Y)$. Therefore, in order to find the minimal separating set for $X$ and $Y$ in $G$, it is suffice to find the minimal separating set for $\alpha_X$ and $\alpha_Y$ in $\text{Aug}(G : \alpha_X, \alpha_Y)$. So, we have reduced this problem to one of separation for single nodes, which can be solved using Algorithm 15.
Shen and Liang in (Shen and Liang, 1997) presents an efficient algorithm for enumerating all minimal \((X, Y)\)-separators, separating given non-adjacent vertices \(X\) and \(Y\) in an undirected connected simple graph \(G = (V, E)\). This algorithm requires \(O(n^3 R_{XY})\) time, where \(|V| = n\) and \(R_{XY}\) is the number of minimal \((X, Y)\)-separators. The algorithm can be generalized for enumerating all minimal \((X, Y)\)-separators that separate non-adjacent vertex sets \(X, Y \subseteq V\), and it requires \(O(n^2(n - n_X - n_Y)R_{XY})\) time. In this case, \(|X| = n_X, |Y| = n_Y\), and \(R_{XY}\) is the number of all minimal \((X, Y)\)-separators. According to Theorem 4.6, using this algorithm for \((G_{ant(X\cup Y)})^a\) solves Problem 18.

**Remark 4.** Since DAGs (directed acyclic graphs) are subclass of AMP chain graphs, one can use the same technique to enumerate all minimal separators in DAGs.

### 4.4 Order-Independent Structure Learning of AMP Chain Graphs

In this section, we explain the original PC-like algorithm proposed in (Peña, 2012) briefly, and we show that this version of the PC-like algorithm is order-dependent, in the sense that the output can depend on the order in which the variables are given. We propose a modification of the PC-like algorithm that removes (part or) all of this order-dependence.

#### 4.4.1 Order-Dependent PC-like Algorithm

The PC-like algorithm for learning AMP CGs under the faithfulness assumption proposed in (Peña, 2012) is formally described in Algorithm 17.

In applications, we do not have perfect conditional independence information. Instead, we assume that we have an i.i.d. sample of size \(n\) of \(V = (X_1, \ldots, X_p)\). In the PC-like algorithm (Peña, 2012) all conditional independence queries are estimated by statistical conditional independence tests at some pre-specified significance level (p.value) \(\alpha\). For example, if the distribution of \(V\) is multivariate Gaussian, one can test for zero partial correlation, see, e.g., (Kalisch and Bühlmann, 2007). For this purpose, we used the `gaussCItest()`
Algorithm 17: The order-dependent PC-like algorithm for learning AMP chain graphs (Peña, 2012)

**Input:** A set $V$ of nodes and a probability distribution $p$ faithful to an unknown AMP CG $G$ and an ordering $\text{order}(V)$ on the variables.

**Output:** A CG $H$ that is triplex equivalent to $G$.

Let $H$ denote the complete undirected graph over $V$;

/* Skeleton Recovery */

for $i \leftarrow 0$ to $|V_H| - 2$ do

while possible do

Select any ordered pair of nodes $u$ and $v$ in $H$ such that $u \in \text{ad}_H(v)$ and $|[\text{ad}_H(u) \cup \text{ad}_H(\text{ad}_H(u))] \setminus \{u, v\}| \geq i$, using $\text{order}(V)$;

/* $\text{ad}_H(x) := \{y \in V | x \rightarrow y, y \rightarrow x, \text{or } x \rightarrow y\}$ */

if there exists $S \subseteq ([\text{ad}_H(u) \cup \text{ad}_H(\text{ad}_H(u))] \setminus \{u, v\})$ s.t. $|S| = i$ and $u \perp \perp v|S$ (i.e., $u$ is independent of $v$ given $S$ in the probability distribution $p$) then

Set $S_{uv} = S_{vu} = S$;

Remove the edge $u \rightarrow v$ from $H$;

end

end

/* Orientation phase: */

while possible do

Apply the rules R1-R4 in the Figure 4.5 to $H$.

end

Replace every edge $\leftarrow \rightarrow (\rightarrow \leftarrow)$ in $H$ with $\rightarrow (\leftarrow)$;

function from the R package pcalg throughout this paper. Let $\text{order}(V)$ denote an ordering on the variables in $V$. We now consider the role of $\text{order}(V)$ in every step of the algorithm.

In the skeleton recovery phase of the PC-like algorithm (Peña, 2012; Peña and Gómez-Olmedo, 2016), the order of variables affects the estimation of the skeleton and the separating sets. In particular, at each level of $i$, the order of variables determines the order in which pairs of adjacent vertices and subsets $S$ of their adjacency sets are considered (see lines 4 and 5 in Algorithm 17). The skeleton $H$ is updated after each edge removal. Hence, the adjacency sets typically change within one level of $i$, and this affects which other conditional independencies are checked, since the algorithm only conditions on subsets of the adjacency sets. When we have perfect conditional independence information, all orderings on the variables lead to the same output. In the sample version, however, we typically
make mistakes in keeping or removing edges. In such cases, the resulting changes in the adjacency sets can lead to different skeletons, as illustrated in Example 16.

Moreover, different variable orderings can lead to different separating sets in the skeleton recovery phase. When we have perfect conditional independence information, this is not important, because any valid separating set leads to the correct triplex decision in the orientation phase. In the sample version, however, different separating sets in the skeleton recovery phase of the algorithm may yield different decisions about triplexes in the orientation phase. This is illustrated in Example 17. The examples were encountered when testing the PC-like algorithm by generating synthesized samples from the DAGs in Figure 4.6(a) and 4.7(a).

**Example 16** (Order-dependent skeleton of the PC-like algorithm.). Suppose that the distribution of $V = \{a, b, c, d, e\}$ is faithful to the DAG in Figure 4.6(a). This DAG encodes the following conditional independencies with minimal separating sets: $b \indep c | a$ and $a \indep e | \{b, c, d\}$.

Suppose that we have an i.i.d. sample of $(a, b, c, d, e)$, and that the following conditional independencies with minimal separating sets are judged to hold at some significance level $\alpha$: $b \indep c | a$, $a \indep e | d, a \indep b | d$, $a \indep c | d$, $b \indep d | e$, and $c \indep d | e$. Thus, the first conditional
independence relation is correct, while the rest of them are false.

We now apply the skeleton recovery phase of the PC-like algorithm with two different orderings: order_1(V) = (d, c, b, a, e) and order_2(V) = (d, e, a, c, b). The resulting skeletons are shown in Figures 4.6(b) and 4.6(c), respectively.

![Figure 4.6: (a) The DAG G, (b) the skeleton returned by Algorithm 17 with order_1(V), (c) the skeleton returned by Algorithm 17 with order_2(V).](image)

We see that the skeletons are different, and that both are incorrect as the edges a — b, a — c, b — d, and c — d are missing. The skeleton for order_2(V) contains an additional error, as there is an additional edge b — c. We now go through Algorithm 17 to see what happened. We start with a complete undirected graph on V. When i = 0, variables are tested for marginal independence, and the algorithm correctly does not remove any edge. When i = 1, there are six pairs of vertices that are thought to be conditionally independent given a subset of size one. Table 4.1 shows the trace table of Algorithm 17 for i = 1 and order_1(V) = (d, c, b, a, e).

Table 4.2 shows the trace table of Algorithm 17 for i = 1 and order_2(V) = (d, e, a, c, b).

No conditional independency is found when i = 2.

**Example 17** (Order-dependent separating sets and triplexes of the PC-like algorithm.). Suppose that the distribution of V = {a, b, c, d, e} is faithful to the DAG in Figure 4.7(a).
This DAG encodes the following conditional independencies with minimal separating sets: 
\( a \perp d | b, a \perp e | (b, c), a \perp e | (c, d), b \perp c, b \perp e | d, \) and \( c \perp d. \)

Suppose that we have an i.i.d. sample of \( (a, b, c, d, e) \). Assume that all true conditional independencies are judged to hold except \( c \perp d \). Suppose that \( c \perp d | b \) and \( c \perp d | e \) are thought to hold. Thus, the first is correct, while the second is false. We now apply the orientation phase of the PC-like algorithm with two different orderings: \( \text{order}_1(V) = (d, c, b, a, e) \) and \( \text{order}_2(V) = (c, d, e, a, b) \). The resulting CGs are shown in Figures 4.7(b) and 4.7(c), respectively. Note that while the separating set for vertices \( c \) and \( d \) with order \( \text{order}_1(V) \) is \( S_{cd} = S_{dc} = \{b\} \), the separating set for them with order \( \text{order}_2(V) \) is \( S_{cd} = S_{dc} = \{e\} \).

This illustrates that order-dependent separating sets in the skeleton recovery phase of
the sample version of the PC-algorithm can lead to order-dependent triplexes in the orientation phase of the algorithm.

4.4.2 Order-Independent (Stable) PC-like Algorithm

We now propose several modifications of the original PC-like algorithm (and hence also of the related algorithms) that remove the order-dependence in the various stages of the algorithm, analogously to what (Colombo and Maathuis, 2014) did for the original PC algorithm in the case of DAGs. For this purpose, we discuss the skeleton and the orientation rules, respectively.

We first consider estimation of the skeleton in the adjacency search (skeleton recovery phase) of the PC-like algorithm. The pseudocode for our modification is given in Algorithm 18. The resulting PC-like algorithm in Algorithm 18 is called stable PC-like.

The main difference between Algorithms 17 and 18 is given by the for-loop on lines 3-5 in the latter one, which computes and stores the adjacency sets $a_H(v_i)$ of all variables after each new size $i$ of the conditioning sets. These stored adjacency sets $a_H(v_i)$ are used whenever we search for conditioning sets of this given size $i$. Consequently, an edge deletion on line 10 no longer affects which conditional independencies are checked for other pairs of variables at this level of $i$. 
Algorithm 18: The order-independent (stable) PC-like algorithm for learning AMP CGs

**Input:** A set $V$ of nodes and a probability distribution $p$ faithful to an unknown AMP CG $G$ and an ordering order($V$) on the variables.

**Output:** A CG $H$ that is triplex equivalent to $G$.

1. Let $H$ denote the complete undirected graph over $V = \{v_1, \ldots, v_n\}$; /* Skeleton Recovery */
2. for $i \leftarrow 0$ to $|V_H| - 2$ do
3.   for $j \leftarrow 1$ to $|V_H|$ do
4.     Set $a_H(v_i) = ad_H(v_i) \cup ad_H(ad_H(v_i))$; /* $ad_H(x) := \{y \in V|x \rightarrow y, y \rightarrow x, \text{ or } x \equiv y\} */
5.   end
6. while possible do
7.     Select any ordered pair of nodes $u$ and $v$ in $H$ such that $u \in ad_H(v)$ and $|a_H(u) \setminus \{u, v\}| \geq i$, using order($V$);
8.     if there exists $S \subseteq (a_H(u) \setminus \{u, v\})$ s.t. $|S| = i$ and $u \perp \!\!\!\!\perp v | S$ (i.e., $u$ is independent of $v$ given $S$ in the probability distribution $p$) then
9.       Set $S_{uv} = S_{vu} = S$;
10.      Remove the edge $u \rightarrow v$ from $H$;
11. end
12. end
13. /* Orientation phase: */
14. while possible do
15.     Apply the rules R1-R4 in the Figure 4.5 to $H$.
16. end
17. Replace every edge $\leftarrow (\rightarrow)$ in $H$ with $\rightarrow (\rightarrow)$;

In other words, at each level of $i$, Algorithm 18 records which edges should be removed, but for the purpose of the adjacency sets it removes these edges only when it goes to the next value of $i$. Besides resolving the order-dependence in the estimation of the skeleton, our algorithm has the advantage that it is easily parallelizable at each level of $i$. The stable PC-like algorithm is correct, i.e. it returns an AMP CG the given probability distribution is faithful to (Theorem 4.9), and yields order-independent skeletons in the sample version (Theorem 4.10). We illustrate the algorithm in Example 18.

**Theorem 4.9.** Let the distribution of $V$ be faithful to an AMP CG $G$, and assume that we are given perfect conditional independence information about all pairs of variables $(u, v)$
in $V$ given subsets $S \subseteq V \setminus \{u, v\}$. Then the output of the stable PC-like algorithm is an AMP CG that is Markov equivalent with $G$.

Proof. The proof of Theorem 4.9 is completely analogous to the proof of Theorem 1 for the original PC-like algorithm in (Peña, 2012). □

Theorem 4.10. The skeleton resulting from the sample version of the stable PC-like algorithm is order-independent.

Proof. We consider the removal or retention of an arbitrary edge $u \rightarrow v$ at some level $i$. The ordering of the variables determines the order in which the edges (line 7 of Algorithm 18) and the subsets $S$ of $a_H(u)$ and $a_H(v)$ (line 8 of Algorithm 18) are considered. By construction, however, the order in which edges are considered does not affect the sets $a_H(u)$ and $a_H(v)$.

If there is at least one subset $S$ of $a_H(u)$ or $a_H(v)$ such that $u \perp \perp v|S$, then any ordering of the variables will find a separating set for $u$ and $v$ (but different orderings may lead to different separating sets as illustrated in Example 17). Conversely, if there is no subset $S'$ of $a_H(u)$ or $a_H(v)$ such that $u \perp \perp v|S'$, then no ordering will find a separating set.

Hence, any ordering of the variables leads to the same edge deletions, and therefore to the same skeleton. □

Example 18 (Order-independent skeletons). We go back to Example 16, and consider the sample version of Algorithm 18. The algorithm now outputs the skeleton shown in Figure 4.6(b) for both orderings $\text{order}_1(V)$ and $\text{order}_2(V)$.

We again go through the algorithm step by step. We start with a complete undirected graph on $V$. No conditional independence found when $i = 0$. When $i = 1$, the algorithm first computes the new adjacency sets: $a_H(v) = V \setminus \{v\}, \forall v \in V$. There are six pairs of variables that are thought to be conditionally independent given a subset of size 1 (see Table 4.3). Since the sets $a_H(v)$ are not updated after edge removals, it does not matter in which order we consider the ordered pairs. Any ordering leads to the removal of six edges.
Table 4.3: The trace table of Algorithm 18 for $i = 1$, order$_1(V) = (d, c, b, a, e)$, and order$_2(V) = (d, e, a, c, b)$. For simplicity, we define $ADJ_H(u) := [ad_H(u) \cup ad_H(ad_H(u))] \setminus \{u, v\}.$

<table>
<thead>
<tr>
<th>Ordered Pair $(u, v)$</th>
<th>$ADJ_H(u)$</th>
<th>$S_{uv}$</th>
<th>Is $S_{uv} \subseteq ADJ_H(u)$?</th>
<th>Is $u \leftarrow v$ removed?</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(d, c)$</td>
<td>$[a, b, e]$</td>
<td>$[e]$</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>$(d, b)$</td>
<td>$[a, c, e]$</td>
<td>$[e]$</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>$(c, b)$</td>
<td>$[a, d, e]$</td>
<td>$[a]$</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>$(c, a)$</td>
<td>$[b, d, e]$</td>
<td>$[d]$</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>$(b, a)$</td>
<td>$[c, d, e]$</td>
<td>$[d]$</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>$(a, e)$</td>
<td>$[b, c, d]$</td>
<td>$[d]$</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Now, we propose a method to resolve the order-dependence in the determination of the triplexes, using an approach similar to that proposed in (Ramsey, Spirtes, and Zhang, 2006).

The **Conservative PC-like algorithm (CPC-like algorithm)** works as follows. Let $H$ be the undirected graph resulting from the skeleton recovery phase of the PC-like algorithm (Algorithm 17). For all unshielded triples $(X_i, X_j, X_k)$ in $H$, determine all subsets $S$ of $ad_H(X_i) \cup ad_H(ad_H(X_i))$ and of $ad_H(X_k) \cup ad_H(ad_H(X_k))$ that make $X_i$ and $X_k$ conditionally independent, i.e., that satisfy $X_i \perp \perp X_k | S$. We refer to such sets as separating sets. The triple $(X_i, X_j, X_k)$ is labelled as unambiguous if at least one such separating set is found and either $X_j$ is in all separating sets or in none of them; otherwise it is labelled as ambiguous. If the triple is unambiguous, it is labeled and then oriented as described in Algorithm 17. So, the orientation rules are adapted so that only unambiguous triples are oriented.

We refer to the combination of the stable PC-like and CPC-like algorithms as the **stable CPC-like algorithm**.

**Theorem 4.11.** Let the distribution of $V$ be faithful to an AMP CG $G$, and assume that we are given perfect conditional independence information about all pairs of variables $(u, v)$ in $V$ given subsets $S \subseteq V \setminus \{u, v\}$. Then the output of the (stable) CPC-like algorithm is an AMP CG that is Markov equivalent with $G$. 
Proof. The skeleton of the learned CG is correct by Theorem 4.9. Now, we prove that for any unshielded triple \((X_i, X_j, X_k)\) in an AMP CG \(G\), \(X_j\) is either in all sets that \(p\)-separate \(X_i\) and \(X_k\) or in none of them. Since \(X_i, X_k\) are not adjacent, they are \(p\)-separated given some subset \(S \setminus \{X_i, X_k\}\) (see Algorithm 15). Based on the pathwise \(p\)-separation criterion for AMP CGs (see Definition 4.2), \(X_j\) is a triplex node in \(G\) if and only if \(X_j \notin \text{An}(S)\). So, \(X_j \notin S\). On the other hand, if \(X_j\) is a non-triplex node then \(X_j \in S\), for all \(S\) that \(p\)-separate \(X_i\) and \(X_k\). Because in this case, \(X_j \in \text{Co}(\text{An}(X_i \cup X_k \cup S))\) and so there is an undirected path \(X_i \rightarrow X_j \rightarrow X_k\) in \((G[X_i \cup X_k \cup S])^p\). Any set \(S \setminus \{X_i, X_k\}\) that does not contain \(X_j\) will fail to \(p\)-separate \(X_i\) and \(X_k\) because of this undirected path. As a result, unshielded triples are all unambiguous. Since all unshielded triples are unambiguous, the orientation rules are as in the original (stable) PC-like algorithm. Therefore, the output of the stable CPC-like algorithm is an AMP CG that is Markov equivalent with \(G\).

\(\square\)

**Theorem 4.12.** The decisions about triplexes in the sample version of the stable CPC-like algorithm is order-independent.

**Proof.** The stable CPC-like algorithm have order-independent skeleton, by Theorem 4.10. In particular, this means that their unshielded triples and adjacency sets are order-independent. The decision about whether an unshielded triple is unambiguous and/or a triplex is based on the adjacency sets of nodes in the triple, which are order independent.

\(\square\)

**Example 19** (Order-independent decisions about triplexes). We consider the sample versions of the stable CPC-like algorithm, using the same input as in Example 17. In particular, we assume that all conditional independencies induced by the AMP CG in Figure 4.7(a) are judged to hold except \(c \perp \perp d\). Suppose that \(c \perp d|b\) and \(c \perp d|e\) are thought to hold.

Denote the skeleton after the skeleton recovery phase by \(H\). We consider the unshielded triple \((c, e, d)\). First, we compute \(a_H(c) = \{a, b, d, e\}\) and \(a_H(d) = \{a, b, c, e\}\). We now consider all subsets \(S\) of these adjacency sets, and check whether \(c \perp d|S\). The following separating sets are found: \(\{b\}\), \(\{e\}\), and \(\{b, e\}\). Since \(e\) is in some but not all of these sepa-
rating sets, the stable CPC-like algorithm determines that the triple is ambiguous, and no orientations are performed. The output of the algorithm is given in Figure 4.7(c).

At this point it should be clear why the modified PC-like algorithm is labeled “conservative”: it is more cautious than the (stable) PC-like algorithm in drawing unambiguous conclusions about orientations. As we showed in Example 19, the output of the (stable) CPC-like algorithm may not be triplex equivalent with the true AMP CG $G$, if the resulting CG contains an ambiguous triple.

Table 4.4 summarizes all order-dependence issues explained above and the corresponding modifications of the PC-like algorithm that removes the given order-dependence problem.

Table 4.4: Order-dependence issues and corresponding modifications of the PC-like algorithm that remove the problem. “Yes” indicates that the corresponding aspect of the graph is estimated order-independently in the sample version.

<table>
<thead>
<tr>
<th>skeleton</th>
<th>triplexes decisions</th>
<th>edges orientations</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC-like</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>stable PC-like</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>stable CPC-like</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

4.5 A DECOMPOSITION-BASED ALGORITHM FOR LEARNING THE STRUCTURE OF AMP CGs

Recently, Javidian and Valtorta (Javidian and Valtorta, 2019a) have developed an algorithm for learning MVR chain graphs that uses a decomposition-based approach. An observation similar to Theorem 4.8 (Javidian and Valtorta, 2018a, Theorem 3) has been used to design the main algorithm in (Javidian and Valtorta, 2019a, Algorithm 2). This algorithm not only reduces complexity and increases the power of computational independence tests but also achieves a better quality with respect to the learned structure. Similarly, we use some of our findings regarding minimal separators in AMP CGs (section 4.3) to design an efficient algorithm for learning AMP chain graphs. Our decomposition based algorithm is a nat-
Figure 4.8: The procedure for learning the structure of the largest deflagged AMP CG from a faithful distribution.

ural extension of the algorithm in (Xie, Zheng, and Zhao, 2006). In particular, the rule in (Xie, Zheng, and Zhao, 2006) for combining local structures into a global skeleton is still applicable. Then orientation rules in (Peña and Gómez-Olmedo, 2016) are used for identifying triplexes. The results of the experiments show that our decomposition based algorithm consistently outperforms (stable) PC-like algorithm. Code for reproducing our results is available at https://github.com/majavid/AMPCGs2019.

Below we briefly list some of the most central concepts used in this section.

Let $\bar{G}_V = (V, \bar{E}_V)$ denote an undirected graph where $\bar{E}_V$ is a set of undirected edges. An undirected edge between two vertices $u$ and $v$ is denoted by $(u, v)$. For a subset $A$ of $V$, let $\bar{G}_A = (A, \bar{E}_A)$ be the subgraph induced by $A$ and $\bar{E}_A = \{e \in \bar{E}_V | e \in A \times A\} = \bar{E}_V \cap (A \times A)$. An undirected graph is called complete if any pair of vertices is connected by an edge. For an undirected graph, we say that vertices $u$ and $v$ are separated by a set of vertices $Z$ if each path between $u$ and $v$ passes through $Z$. We say that two distinct vertex sets $X$ and $Y$ are separated by $Z$ if and only if $Z$ separates every pair of vertices $u$ and $v$ for any $u \in X$ and $v \in Y$. We say that an undirected graph $\bar{G}_V$ is an undirected independence graph (UIG) for
CG $G$ if the fact that a set $Z$ separates $X$ and $Y$ in $\tilde{G}_V$ implies that $Z$ $p$-separates $X$ and $Y$ in $G$. Note that the augmented graph derived from CG $G$, $(G)^a$, is an undirected independence graph for $G$. We say that $\tilde{G}_V$ can be decomposed into subgraphs $\tilde{G}_A$ and $\tilde{G}_B$ if

1. $A \cup B = V$, and
2. $C = A \cap B$ separates $V \setminus A$ and $V \setminus B$ in $\tilde{G}_V$.

The above decomposition does not require that the separator $C$ be complete, which is required for weak decomposition defined in (Lauritzen, 1996). In this section, we show that a problem of learning the structure of CG can also be decomposed into problems for its decomposed subgraphs even if the separator is not complete.

A *triangulated (chordal)* graph is an undirected graph in which all cycles of four or more vertices have a chord, which is an edge that is not part of the cycle but connects two vertices of the cycle (see, for example, Figure 4.9). For an undirected graph $\tilde{G}_V$ which is not triangulated, we can add extra (“fill-in”) edges to it such that it becomes a triangulated graph, denoted by $\tilde{G}_V^t$.

In this section, we assume that all independencies of a probability distribution of variables in $V$ can be checked by $p$-separations of $G$, called the faithfulness assumption (Spirtes, Glymour, and Scheines, 2000). The *faithfulness assumption* means that all independencies and conditional independencies among variables can be represented by $G$.
The global skeleton is an undirected graph obtained by dropping direction of CG. A
local skeleton for a subset \( A \) of variables is an undirected subgraph for \( A \) in which the ab-
sence of an edge \((u, v)\) implies that there is a subset \( S \) of \( A \) such that \( u \perp \perp v|S \). Now, we
introduce the notion of \( p\)-separation trees, which is used to facilitate the representation of
the decomposition. The concept is similar to the junction tree of cliques and the indepen-
dence tree introduced for DAGs as \( d\)-separation trees in (Xie, Zheng, and Zhao, 2006). Let
\( C = \{C_1, \ldots, C_H\} \) be a collection of distinct variable sets such that for \( h = 1, \ldots, H, C_h \subseteq V \).
Let \( T \) be a tree where each node corresponds to a distinct variable set in \( C \), to be displayed
as an oval (see, for example, Figure 4.10). An undirected edge \( e = (C_i, C_j) \) connecting
nodes \( C_i \) and \( C_j \) in \( T \) is labeled with a separator \( S = C_i \cap C_j \), which is displayed as a
rectangle. Removing an edge \( e \) or, equivalently, removing a separator \( S \) from \( T \) splits \( T \)
into two subtrees \( T_1 \) and \( T_2 \) with node sets \( C_1 \) and \( C_2 \) respectively. We use \( V_i \) to denote the
union of the vertices contained in the nodes of the subtree \( T_i \) for \( i = 1, 2 \).

![Figure 4.10: The \( p\)-separation tree of CG \( G \) in Figure 4.9](image)

Notice that a separator is defined in terms of a tree whose nodes consist of variable
sets, while the \( p\)-separator is defined based on chain graph. In general, these two concepts
are not related, though for a \( p\)-separation tree its separator must be some corresponding
\( p\)-separator in the underlying AMP chain graph. The definition of \( p\)-separation trees for
AMP chain graphs is similar to that of junction trees of cliques, see (Cowell et al., 1999; Lauritzen, 1996). Actually, it is not difficult to see that a junction tree of chain graph $G$ is also a $p$-separation tree. However, as in (Ma, Xie, and Geng, 2008), we point out two differences here: (a) a $p$-separation tree is defined with $p$-separation and it does not require that every node be a clique or that every separator be complete on the augmented graph; (b) junction trees are mostly used in inference engines, while our interest in $p$-separation trees is mainly derived from their power in facilitating the decomposition of structural learning.

4.5.1 CONSTRUCTING A $p$-SEPARATION TREE FROM OBSERVED DATA

As proposed in (Xie, Zheng, and Zhao, 2006), one can construct a $d$-separation tree from observed data. In this section, we extend Theorem 2 of (Xie, Zheng, and Zhao, 2006), which guarantees that their method for constructing a separation tree from data is valid for AMP chain graphs. To construct an undirected independence graph in which the absence of an edge $(u, v)$ implies $u \perp \perp v | V \setminus \{u, v\}$, we can start with a complete undirected graph, and then for each pair of variables $u$ and $v$, an undirected edge $(u, v)$ is removed if $u$ and $v$ are independent conditional on the set of all other variables (Xie, Zheng, and Zhao, 2006). For normally distributed data, the undirected independence graph can be efficiently constructed by removing an edge $(u, v)$ if and only if the corresponding entry in the concentration matrix (inverse covariance matrix) is zero (Lauritzen, 1996, Proposition 5.2). For this purpose, performing a conditional independence test for each pair of random variables using the partial correlation coefficient can be used. If the $p$-value of the test is smaller than the given threshold, then there will be an edge on the output graph. For discrete data, a test of conditional independence given a large number of discrete variables may be of extremely low power. To cope with such difficulty, a local discovery algorithm called Max-Min Parents and Children (MMPC) (Tsamardinos, Aliferis, and Statnikov, 2003) or the forward selection procedure described in (Edwards, 2000) can be applied.
Theorem 4.13. A junction tree constructed from an undirected independence graph for AMP CG \( G \) is a \( p \)-separation tree for \( G \).

Proof. See Appendix D. □

A \( p \)-separation tree \( T \) only requires that all \( p \)-separation properties of \( T \) also hold for AMP CG \( G \), but the reverse is not required. Thus we only need to construct an undirected independence graph that may have fewer conditional independencies than the augmented graph, and this means that the undirected independence graph may have extra edges added to the augmented graph. As (Xie, Zheng, and Zhao, 2006) observe for \( d \)-separation in DAGs, if all nodes of a \( p \)-separation tree contain only a few variables, “the null hypothesis of the absence of an undirected edge may be tested statistically at a larger significance level.”

Since there are standard algorithms for constructing junction trees from UIGs (Cowell et al., 1999, Chapter 4, Section 4), the construction of separation trees reduces to the construction of UIGs. In this sense, Theorem 4.13 enables us to exploit various techniques for learning UIGs to serve our purpose. More suggested methods for learning UIGs from data, in addition to the above mentioned techniques, can be found in (Ma, Xie, and Geng, 2008).

Example 20. To construct a \( p \)-separation tree for the AMP CG \( G \) in Figure 4.9(a), at first an undirected independence graph is constructed by starting with a complete graph and removing an edge \((u, v)\) if \( u \perp \perp v | V \setminus \{u, v\} \). An undirected graph obtained in this way is the augmented graph of AMP CG \( G \). In fact, we only need to construct an undirected independence graph which may have extra edges added to the augmented graph. Next triangulate the undirected graph and finally obtain the \( p \)-separation tree, as shown in Figure 4.9(c) and Figure 4.10 respectively.

4.5.2 Structural Learning by Decomposition

Applying the following theorem to structural learning, we can split a problem of searching for \( p \)-separators and building the skeleton of a CG into small problems for every node of
Theorem 4.14. Let $T$ be a $p$-separation tree for AMP CG $G$ and $u$ and $v$ be two vertices that do not belong to the same chain component. So, vertices $u$ and $v$ are $p$-separated by $S \subseteq V$ in $G$ if and only if (i) $u$ and $v$ are not contained together in any node $C$ of $T$ or (ii) there exists a node $C$ that contains both $u$ and $v$ such that a subset $S'$ of $C$ $p$-separates $u$ and $v$.

Proof. See Appendix D. □

According to Theorem 4.14, a problem of searching for a $p$-separator $S$ of $u$ and $v$ in all possible subsets of $V$ is localized to all possible subsets of nodes in a $p$-separation tree that contain $u$ and $v$. For a given $p$-separation tree $T$ with the node set $C = \{C_1, \ldots, C_H\}$, we can recover the skeleton and all triplexes for an AMP CG as follows. First we construct a local skeleton for every node $C_h$ of $T$, which is constructed by starting with a complete undirected subgraph and removing an undirected edge $(u, v)$ if there is a subset $S$ of $C_h$ such that $u$ and $v$ are independent conditional on $S$. For this purpose, we can use the PC-like algorithm in (Peña, 2012) or the stable PC-like algorithm (Algorithm 18). Then, in order to construct the global skeleton, we first combine all these local skeletons together. Note that it is possible that some edges that are present in some local skeletons may be absent in other local skeletons. Also, two non-adjacent vertices $u$ and $v$ in the AMP CG $G$ that belong to the same chain component may be adjacent in the temporary global skeleton. (Note that Theorem 4.14 only guarantees the existence of the $p$- separators for those non-adjacent vertices that do not belong to the same chain component. In Appendix D, we provide an example that shows that Theorem 4.14 cannot be strengthened.) In order to get rid of the extra edges in the resulting undirected graph, we apply a removal procedure that is similar to the skeleton recovery phase of the PC-like algorithm. However, instead of the complete undirected graph we use the resulting undirected graph obtained in the previous
step. Then we orient undirected edges using rules R1-R4 in Figure 4.5 (Peña, 2012; Peña and Gómez-Olmedo, 2016). This process is formally described in Algorithm 19.

We prove that the global skeleton and all triplexes obtained by applying the decomposition in Algorithm 19 are correct, that is, they are the same as those obtained from the joint distribution of V; see Appendix D for proof details. Note that separators in a p-separation tree may not be complete in the augmented graph. Thus the decomposition is weaker than the decomposition usually defined for parameter estimation (Cowell et al., 1999; Lauritzen, 1996).

Remark 5. One can apply Algorithm 3 in (Roverato and Rocca, 2006) to the resulting chain graph of Algorithm 19 to obtain the largest deflagged graph. Also, one can apply Algorithm 1 in (Sonntag and Peña, 2015a) to the resulting chain graph of Algorithm 19 to obtain the AMP essential graph.

4.5.3 Complexity Analysis and Advantages

In this section, we start by comparing our algorithm with the main algorithm in (Xie, Zheng, and Zhao, 2006) that is designed specifically for DAG structural learning when the underlying graph structure is a DAG. We make this choice of the DAG specific algorithm so that both algorithms can have the same separation tree as input and hence are directly comparable.

The same advantages mentioned by (Xie, Zheng, and Zhao, 2006) for their BN structural learning algorithm hold for our algorithm when applied to AMP CGs. For the reader’s convenience, we list them here. First, by using the p-separation tree, independence tests are performed only conditionally on smaller sets contained in a node of the p-separation tree rather than on the full set of all other variables. Thus our algorithm has higher power for statistical tests. Second, the computational complexity can be reduced. The number of conditional independence tests for constructing the equivalence class is used as characteristic operation for this complexity analysis. Decomposition of graphs is a computationally
Algorithm 19: A decomposition-based recovery algorithm for AMP chain graphs

**Input:** a probability distribution $p$ faithful to an unknown AMP CG $G$.

**Output:** the largest deflagged graph corresponding to the AMP CG $G$.

1. Construct a $p$-separation tree $T$ with a node set $C = \{C_1, \ldots, C_I\}$ as discussed in Section 4.5.1;
2. Set $S = \emptyset$;
   
   /* Local skeleton recovery: */
   
3. for $i \leftarrow 1$ to $I$ do
   
4. Start from a complete undirected graph $\tilde{G}_i$ with vertex set $C_i$;
5. for each vertex pair \{u, v\} $\subseteq C_i$ do
6. if $\exists S_{uv} \subseteq C_i$ such that $u \perp \perp v|S_{uv}$ then
7. Delete the edge (u, v) in $\tilde{G}_i$;
8. Add $S_{uv}$ to $S$;
9. end
10. end
11. end
   
/* Global skeleton recovery: */

12. Initialize the edge set $\tilde{E}_V$ of $\tilde{G}_V$ as the union of all edge sets of $\tilde{G}_i, i = 1, \ldots, I$;
13. Set $H = \tilde{G}_V$;
14. for $i \leftarrow 0$ to $|V_H| - 2$ do
15. while possible do
16. Select any ordered pair of nodes $u$ and $v$ in $H$ such that $u \in \text{ad}_H(v)$ and $|\text{ad}_H(u) \cup \text{ad}_H(\text{ad}_H(u))\} \setminus \{u, v\}| \geq i$;
17. if there exists $S \subseteq \{(\text{ad}_H(u) \cup \text{ad}_H(\text{ad}_H(u))) \setminus \{u, v\}\}$ s.t. $|S| = i$ and $u \perp \perp v|S$ (i.e., $u$ is independent of $v$ given $S$ in the probability distribution $p$) then
18. Set $S_{uv} = S_{vu} = S$;
19. Remove the edge $u \leftarrow v$ from $H$;
20. end
21. end
22. end
   
/* Orientation phase (Peña, 2012): */
23. while possible do
24. Apply the rules R1-R4 in the Figure 4.5 to $H$.
   
/* A block is represented by a perpendicular line at the edge end such as in \| or $\perp$, and it means that the edge cannot be a directed edge pointing in the direction of the block. Note that $\perp$ means that the edge must be undirected. The ends of some of the edges in the rules are labeled with a circle such as in $\bigcirc$ or $\circ$. The circle represents an unspecified end, i.e. a block or nothing. */
25. end
26. Replace every edge $u \leftarrow (u \leftarrow i)$ in $H$ with $\rightarrow (\rightarrow)$.
simple task compared to the task of testing conditional independence for a large number of triples of sets of variables. The triangulation of an undirected graph is used in our algorithms to construct a \( p \)-separation tree from an undirected independence graph. Although the problem for optimally triangulating an undirected graph is \( \text{NP-hard} \), sub-optimal triangulation methods (Berry et al., 2004) may be used provided that the obtained tree does not contain too large nodes to test conditional independencies. Two of the best known algorithms are lexicographic search and maximum cardinality search, and their complexities are \( O(|V||E|) \) and \( O(|V| + |E|) \), respectively (Berry et al., 2004). Thus in our algorithms, the conditional independence tests dominate the algorithmic complexity.

The complexity of Algorithm 19 is \( O(Hm^22^m) \) as claimed in (Xie, Zheng, and Zhao, 2006, Section 6), where \( H \) is the number of \( p \)-separation tree nodes (usually \( H \ll |V| \)) and \( m = \max_h |C_h| \) where \( |C_h| \) denotes the number of variables in \( C_h \) \((m \text{ usually is much less than } |V|)\).

4.5.4 Evaluation

In this subsection, we evaluate the performance of our algorithms in various setups using simulated / synthetic data sets. We first compare the performance of our algorithm with the (stable) PC-like learning algorithm by running them on randomly generated AMP chain graphs. We then compare our method with the (stable) PC-like algorithm on different discrete Bayesian networks such as ASIA, INSURANCE, ALARM, and HAILFINDER that have been widely used in evaluating the performance of structural learning algorithms.

Empirical simulations show that our algorithm achieves competitive results with the (stable) PC-like learning algorithm; in particular, in the Gaussian case the decomposition-based algorithm outperforms the (stable) PC-like algorithm. Algorithms 19 and the (stable) PC-like algorithm have been implemented in the R language. All the results reported here are based on our R implementation: https://github.com/majavid/AMPCGs2019.
To investigate the performance of the proposed learning methods in this chapter, we use the same approach that (Ma, Xie, and Geng, 2008) used in evaluating the performance of the LCD algorithm on LWF chain graphs. We run our algorithms on randomly generated AMP chain graphs and then we compare the results and report summary error measures in all cases.

**Data Generation Procedure.** First we explain the way in which the random AMP chain graphs and random samples are generated. Given a vertex set \( V \), let \( p = |V| \) and \( N \) denote the average degree of edges (including undirected and pointing out and pointing in) for each vertex. We generate a random AMP chain graph on \( V \) as follows:

- Order the \( p \) vertices and initialize a \( p \times p \) adjacency matrix \( A \) with zeros;
- For each element in the lower triangle part of \( A \), set it to be a random number generated from a Bernoulli distribution with probability of occurrence \( s = N/(p - 1) \);
- Symmetrize \( A \) according to its lower triangle;
- Select an integer \( k \) randomly from \( \{1, \ldots, p\} \) as the number of chain components;
- Split the interval \([1, p]\) into \( k \) equal-length subintervals \( I_1, \ldots, I_k \) so that the set of variables falling into each subinterval \( I_m \) forms a chain component \( C_m \);
- Set \( A_{ij} = 0 \) for any \((i, j)\) pair such that \( i \in I_l, j \in I_m \) with \( l > m \).

This procedure yields an adjacency matrix \( A \) for a chain graph with \( (A_{ij} = A_{ji} = 1) \) representing an undirected edge between \( V_i \) and \( V_j \) and \( (A_{ij} = 1, A_{ji} = 0) \) representing a directed edge from \( V_i \) to \( V_j \). Moreover, it is not difficult to see that \( \mathbb{E}[\text{vertex degree}] = N \), where an adjacent vertex can be linked by either an undirected or a directed edge. In order to sample the artificial CGs, we first transformed them into DAGs and then sampled these DAGs under marginalization and conditioning as indicated in (Peña, 2014b). The
transformation of an AMP CG $G$ into a DAG $H$ is as follows: First, every node $X$ in $G$ gets a new parent $\epsilon^X$ representing an error term, which by definition is never observed. Then, every undirected edge $X \rightarrow Y$ in $G$ is replaced by $\epsilon^X \rightarrow S_{XY} \leftarrow \epsilon^Y$ where $S_{XY}$ denotes a selection bias node, i.e. a node that is always observed. Given a randomly generated chain graph $G$ with ordered chain components $C_1, \ldots, C_k$, we generate a Gaussian distribution on the corresponding transformed DAG $H$ using the Hugin API. Note that the probability distributions of samples are likely to satisfy the faithfulness assumption, but there is no guarantee i.e., samples can have additional independencies that cannot be represented by the CG $G$.

**Experimental Results.** We evaluate the performance of the proposed algorithms in terms of six measurements: (a) the true positive rate (TPR)\(^1\), (b) the false positive rate (FPR)\(^2\), (c) the true discovery rate (TDR)\(^3\), (d) accuracy (ACC) for the skeleton, (e) the structural Hamming distance (SHD)\(^4\), and (f) run-time for the LCG recovery algorithms. In short, $TPR = \frac{\text{true positive (TP)}}{\text{the number of real positive cases in the data (Pos)}}$ is the ratio of the number of correctly identified edges over total number of edges (in true graph), $FPR = \frac{\text{false positive (FP)}}{\text{the number of real negative cases in the data (Neg)}}$ is the ratio of the number of incorrectly identified edges over total number of gaps, $TDR = \frac{\text{true positive (TP)}}{\text{the total number of edges in the recovered CG}}$ is the ratio of the number of correctly identified edges over total number of edges (both in estimated graph), $ACC = \frac{\text{true positive (TP)} + \text{true negative (TN)}}{\text{Pos} + \text{Neg}}$ and $SHD$ is the number of legitimate operations needed to change the current resulting graph to the true CG, where legitimate operations are: (a) add or delete an edge and (b) insert, delete or reverse an edge orientation. In principle, a large TPR and ACC, a small FPR and SHD indicate good performance.

In our simulation, we change three parameters $p$ (the number of vertices), $n$ (sample

---

1 Also known as sensitivity, recall, and hit rate.

2 Also known as fall-out.

3 Also known as precision or positive predictive value.

4 This is the metric described in (Tsamardinos et al., 2006) to compare the structure of the learned and the original graphs.
size) and $N$ (expected number of adjacent vertices) as follows:

- $p \in \{10, 20, 30, 40, 50\}$,
- $n \in \{500, 1000, 5000, 10000\}$, and
- $N \in \{2, 3\}$.

For each $(p, N)$ combination, we first generate 30 random AMP CGs. We then generate a random Gaussian distribution based on each graph and draw an identically independently distributed (i.i.d.) sample of size $n$ from this distribution for each possible $n$. For each sample, three different significance levels ($\alpha = 0.005, 0.01, 0.05$) are used to perform the hypothesis tests. The null hypothesis $H_0$ is “two variables $u$ and $v$ are conditionally independent given a set $C$ of variables” and alternative $H_1$ is that $H_0$ may not hold. We then compare the results to access the influence of the significance testing level on the performance of our algorithms.

Figure 4.11 shows that: (a) as we expected (Ma, Xie, and Geng, 2008; Kalisch and Bühlmann, 2007; Javidian and Valtorta, 2019a; Javidian, Valtorta, and Jamshidi, 2019b), both algorithms work well on sparse graphs ($N = 2, 3$), (b) for both algorithms, typically the TPR, TDR, and ACC increase with sample size, (c) for both algorithms, typically the SHD and FPR decrease with sample size, (d) a large significance level ($\alpha = 0.05$) typically yields large TPR, FPR, and SHD, (e) in almost all cases, the performance of the decomposition-based algorithm based on all error measures i.e., TPR, FPR, TDR, ACC, and SHD is better than the performance of the (stable) PC-like algorithm, (f) in most cases, error measures based on $\alpha = 0.01$ and $\alpha = 0.005$ are very close. Generally, our empirical results suggests that in order to obtain a better performance, we can choose a small value (say $\alpha = 0.005$ or 0.01) for the significance level of individual tests along with large sample (say $n = 5000$ or 10000). However, the optimal value for a desired overall error rate may depend on the sample size, significance level, and the sparsity of the underlying graph, (g) while the stable PC-like algorithm has a better TDR and FPR in comparison with the orig-
inal PC-like algorithm, the original PC-like algorithm has a better TPR as observed in the case of DAGs (Colombo and Maathuis, 2014). This can be explained by the fact that the stable PC-like algorithm tends to perform more tests than the original PC-like algorithm, and (h) there is no meaningful difference between the performance of the stable PC-like algorithm and the original PC-like algorithm in terms of error measures ACC and SHD.

When considering average running times versus sample sizes, as shown in Figures 4.12, we observe that: (a) the average run time increases when sample size increases; (b) the average run times based on $\alpha = 0.01$ and $\alpha = 0.005$ are very close and in all cases better than $\alpha = 0.05$, while choosing $\alpha = 0.005$ yields a consistently (albeit slightly) lower average run time across all the settings; (c) generally, the average run time for the decomposition-based algorithm is better than that for the (stable) PC-like algorithm. In Figure 4.13, the algorithms are compared by counting the number of independence tests, rather than runtime, in order to reduce the impact of different implementations (R packages). We observe that: (a) the average number of independence tests increases when sample size increases; (b) the average number of independence tests based on $\alpha = 0.01$ and $\alpha = 0.005$ are close and in all cases better than $\alpha = 0.05$, while choosing $\alpha = 0.005$ yields a consistently lower average number of independence tests across all the settings; (c) generally, the average number of independence tests for the decomposition-based algorithm is better than that for the (stable) PC-like algorithm. These observations are consistent with the theoretical complexity analysis that we discussed in section 3.5.3. In fact, our findings confirm that the decomposition-based algorithm reduces complexity and increases the power of computational independence tests.

**Performance on Discrete Bayesian Networks**

Bayesian networks are special cases of AMP CGs. It is of interest to see whether the decomposition-based algorithms still work well when the data are actually generated from
Figure 4.11
Figure 4.11: First two columns show the performance of the decomposition based (LCD) and (stable) PC-like algorithms for randomly generated Gaussian chain graph models: average over 30 repetitions with 50 variables correspond to N = 2, 3, and the significance level $\alpha = 0.005$. In each plot, the solid blue line corresponds to the LCD algorithm, the dashed red line corresponds to the original PC-like algorithm, and the dotted grey line corresponds to the stable PC-like algorithm. The third column shows the performance of the decomposition based (LCD) algorithm for randomly generated Gaussian chain graph models: average over 30 repetitions with 50 variables correspond to N = 2, and the significance level $\alpha = 0.05, 0.01, 0.005$. In each plot, the solid blue line corresponds to $\alpha = 0.05$, the dashed red line corresponds to $\alpha = 0.01$, and the dotted grey line corresponds to $\alpha = 0.005$.

Figure 4.12: The first two columns show the running times of the decomposition-based and PC-like algorithms for randomly generated Gaussian chain graph models: average over 30 repetitions with 50 variables correspond to N = 2, 3 and significance levels $\alpha = 0.005$. In each plot, the solid blue line corresponds to the LCD algorithm, the dashed red line corresponds to the original PC-like algorithm, and the dotted grey line corresponds to the stable PC-like algorithm. The third column shows the running times of the decomposition based (LCD) algorithm for randomly generated Gaussian chain graph models: average over 30 repetitions with 50 variables correspond to N = 2, and the significance level $\alpha = 0.05, 0.01, 0.005$. In each plot, the solid blue line corresponds to $\alpha = 0.05$, the dashed red line corresponds to $\alpha = 0.01$, and the dotted grey line corresponds to $\alpha = 0.005$. 

159
Figure 4.13: The first two columns show the number of independence tests used by the decomposition-based and PC-like algorithms for randomly generated Gaussian chain graph models: average over 30 repetitions with 50 variables corresponding to average degrees $N = 2, 3$ and significance level $\alpha = 0.005$. In each plot, the solid blue line corresponds to the LCD algorithm, the dashed red line corresponds to the original PC-like algorithm, and the dotted grey line corresponds to the stable PC-like algorithm. The third column shows the number of independence tests used by the decomposition based (LCD) algorithm for randomly generated Gaussian chain graph models: average over 30 repetitions with 50 variables corresponding to average degree $N = 2$, and significance levels $\alpha = 0.05, 0.01, 0.005$. In each plot, the solid blue line corresponds to $\alpha = 0.05$, the dashed red line corresponds to $\alpha = 0.01$, and the dotted grey line corresponds to $\alpha = 0.005$.

a Bayesian network. For this purpose, we perform simulation studies for four well-known Bayesian networks from Bayesian Network Repository: ASIA, INSURANCE, ALARM, and HAILFINDER. We purposefully selected these networks because they have different sizes (from small to large numbers of nodes, edges, and parameters). We briefly introduce these networks here:

- **ASIA** (Lauritzen and Spiegelhalter, 1988) with 8 nodes, 8 edges, and 18 parameters, it describes the diagnosis of a patient at a chest clinic who may have just come back from a trip to Asia and may be showing dyspnea. Standard learning algorithms are not able to recover the true structure of the network because of the presence of a functional node.

- **INSURANCE** (Binder et al., 1997) with 27 nodes, 52 edges, and 984 parameters, it evaluates car insurance risks.

- **ALARM** (Beinlich et al., 1989) with 37 nodes, 46 edges and 509 parameters, it was designed by medical experts to provide an alarm message system for intensive care
unit patients based on the output a number of vital signs monitoring devices.

- HAILFINDER (Abramson et al., 1996) with 56 nodes, 66 edges, and 2656 parameters, it was designed to forecast severe summer hail in northeastern Colorado.

We compared the performance of our algorithms for these Bayesian networks for three different significance levels ($\alpha = 0.05/0.01/0.005$). The results of all learning methods are summarized in Table 4.5. The results indicate that the performance of both algorithms in terms of FPR, ACC, and SHD are very similar. However, the decomposition-based algorithm outperforms the PC-like algorithm in terms of the TPR.

The Structural Hamming Distance (SHD) directly compares the structure of the learned and the original networks. Although the SHD of the learned CG for ASIA network based on the PC-like algorithm is less than for the LCD algorithm, this does not mean that the quality of the learned CG based on the PC-like algorithm is better than for the LCD algorithm. In fact, as one can see in Figure 4.14, the learned CG by the LCD algorithm is more informative and resembles the ASIA network more than the CG learned by the PC-like algorithm. So, in order to have more accurate evaluation about the output of the different algorithms, (based on the above observation, our suggestion is considering) one should consider all of error measures together.

4.6 Discussion and Conclusion

This chapter addresses two main problems in the context of AMP chain graphs (CGs): finding minimal separators and structure learning. The solution of the first problem is used to design efficient algorithms for the second one.

We first studied and solved the problem of finding minimal separating sets for pairs of variables in an AMP CGs. We also studied some extensions of the basic problem that include include finding a minimal separator from a restricted set of nodes, finding a minimal separator for two given disjoint sets, testing whether a given separator is minimal, and
Table 4.5: Results for discrete samples from the ASIA, INSURANCE, ALARM, and HAILFINDER networks respectively. Each row corresponds to the significance level: $\alpha = 0.05/0.01/0.005$ respectively.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>TPR</th>
<th>FPR</th>
<th>ACC</th>
<th>SHD</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCD Algorithm</td>
<td>0.625</td>
<td>0.2</td>
<td>0.75</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>0.625</td>
<td>0.2</td>
<td>0.75</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>0.625</td>
<td>0.2</td>
<td>0.75</td>
<td>9</td>
</tr>
<tr>
<td>Stable PC-Like Algorithm</td>
<td>0.5</td>
<td>0</td>
<td>0.8571</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>0</td>
<td>0.8571</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>0</td>
<td>0.8571</td>
<td>6</td>
</tr>
<tr>
<td>Original PC-Like Algorithm</td>
<td>0.5</td>
<td>0</td>
<td>0.8571</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>0</td>
<td>0.8571</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>0</td>
<td>0.8571</td>
<td>6</td>
</tr>
<tr>
<td>LCD Algorithm</td>
<td>0.577</td>
<td>0.020</td>
<td>0.920</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>0.577</td>
<td>0.013</td>
<td>0.926</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>0.596</td>
<td>0.010</td>
<td>0.931</td>
<td>34</td>
</tr>
<tr>
<td>Stable PC-Like Algorithm</td>
<td>0.288</td>
<td>0</td>
<td>0.89</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>0.288</td>
<td>0</td>
<td>0.89</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>0.288</td>
<td>0</td>
<td>0.89</td>
<td>48</td>
</tr>
<tr>
<td>Original PC-Like Algorithm</td>
<td>0.385</td>
<td>0</td>
<td>0.909</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>0.385</td>
<td>0</td>
<td>0.909</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>0.385</td>
<td>0</td>
<td>0.909</td>
<td>45</td>
</tr>
<tr>
<td>LCD Algorithm</td>
<td>0.587</td>
<td>0.011</td>
<td>0.961</td>
<td>46</td>
</tr>
<tr>
<td></td>
<td>0.609</td>
<td>0.019</td>
<td>0.955</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td>0.630</td>
<td>0.014</td>
<td>0.96</td>
<td>47</td>
</tr>
<tr>
<td>Stable PC-Like Algorithm</td>
<td>0.348</td>
<td>0</td>
<td>0.955</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>0.348</td>
<td>0</td>
<td>0.955</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>0.348</td>
<td>0</td>
<td>0.955</td>
<td>40</td>
</tr>
<tr>
<td>Original PC-Like Algorithm</td>
<td>0.609</td>
<td>0</td>
<td>0.973</td>
<td>29</td>
</tr>
<tr>
<td></td>
<td>0.609</td>
<td>0</td>
<td>0.973</td>
<td>29</td>
</tr>
<tr>
<td></td>
<td>0.609</td>
<td>0</td>
<td>0.973</td>
<td>29</td>
</tr>
<tr>
<td>LCD Algorithm</td>
<td>0.667</td>
<td>0.001</td>
<td>0.984</td>
<td>53</td>
</tr>
<tr>
<td></td>
<td>0.742</td>
<td>0.007</td>
<td>0.988</td>
<td>54</td>
</tr>
<tr>
<td></td>
<td>0.727</td>
<td>0.001</td>
<td>0.987</td>
<td>50</td>
</tr>
<tr>
<td>Stable PC-Like Algorithm</td>
<td>0.394</td>
<td>0</td>
<td>0.974</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td>0.394</td>
<td>0</td>
<td>0.974</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td>0.394</td>
<td>0</td>
<td>0.974</td>
<td>47</td>
</tr>
<tr>
<td>Original PC-Like Algorithm</td>
<td>0.455</td>
<td>0.005</td>
<td>0.972</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>0.455</td>
<td>0.005</td>
<td>0.971</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>0.455</td>
<td>0.005</td>
<td>0.971</td>
<td>50</td>
</tr>
</tbody>
</table>
This chapter also contains a detailed study of two constraint-based approaches to learning AMP chain graphs (CGs): PC-like and decomposition-based (LCD). We showed that the proposed PC-like algorithm for learning AMP CGs in (Peña, 2012; Peña and Gómez-Olmedo, 2016) is order-dependent, in the sense that the output can depend on the order in which the variables are given. We propose two modifications of the PC-like algorithm that remove part or all of this order-dependence. The PC-like algorithm is a constraint-based algorithm that learns the structure of the underlying AMP chain graph in three steps: (a) determining the skeleton: the resulting undirected graph in this phase contains an undirected edge $u - v$ iff there is no set $S \subseteq V \setminus \{u, v\}$ such that $u \perp \perp v \mid S$; (b) determining triplexes and orienting some of the undirected edges into directed edges according to a set of rules applied iteratively; (c) transforming the resulting graph in the previous step into the essential AMP CG (or largest deflagged graph) after step (b). The decomposition-based algorithm is also a constraint-based algorithm that is based on a divide and conquer approach.

Figure 4.14: Original ASIA network; networks learned using LCD and PC-like algorithms, respectively.
and contains three steps: (a) determining the skeleton by a divide-and-conquer approach; (b) determining triplexes and orienting some of the undirected edges into directed edges according to a set of rules applied iteratively with localized search for \( p \)-separators; continuing with step (c) exactly as in the PC-like algorithm. The correctness of both algorithms lies upon the assumption that the probability distribution \( p \) is faithful to some AMP CG. As for the PC-like algorithms, unless the probability distribution \( p \) of the data is faithful to some AMP CG, the learned CG cannot be ensured to factorize \( p \) properly. Empirical simulations in the Gaussian case show that both algorithms yield good results when the underlying graph is sparse; this holds also in the discrete case, according to experiments with typical Bayesian networks. The decomposition-based algorithm achieves competitive results with the PC-like learning algorithms in both the Gaussian and discrete cases. In fact, the decomposition-based method usually outperforms the PC-like algorithms in all five error measures i.e., TPR, FPR, TDR, ACC, and SHD. Since our implementation of the decomposition-based algorithm is based on the LCD R package, with fixed number of samples, one can expect better performance if we replace the asymptotic test used in the LCD R package with an exact test. However, there is a trade-off between accuracy and computational time. The decomposition-based algorithm exploits our results on separating sets. It exhibits reduced complexity, as measured by run time and number of conditional independence tests, it enhances the power of conditional independence tests by reducing the number of separating sets that need to be considered, and, according to our experimental evaluation, achieves better quality with respect to the learned structure.

The natural continuation of the work presented here would be to develop a learning algorithm with weaker assumptions than the one presented. This could for example be a learning algorithm that only assumes that the probability distribution satisfies the composition property. It should be mentioned that (Peña, Sonntag, and Nielsen, 2014) developed an algorithm for learning LWF CGs under the composition property. However, (Peña, 2014a) proved that the same technique cannot be used for AMP chain graphs. We believe
that our decomposition-based approach is extendable to the structural learning of marginal AMP chain graphs (Peña and Gómez-Olmedo, 2016) and ancestral graphs (Richardson and Spirtes, 2002). So, the natural continuation of the work presented here would be to develop a learning algorithm via decomposition for marginal AMP chain graphs and ancestral graphs under the faithfulness assumption.
Chapter 5

Bayesian Hypergraphs

Probabilistic Graphical Models (PGMs) enjoy a well-deserved popularity because they allow explicit representation of structural constraints in the language of graphs and similar structures. From the perspective of efficient belief update, factorization of the joint probability distribution of random variables corresponding to variables in the graph is paramount, because it allows decomposition of the calculation of the evidence or of the posterior probability (Lauritzen and Jensen, 1997). The proliferation of different PGMs that allow factorizations of different kinds leads us to consider a more general graphical structure in this chapter, namely directed acyclic hypergraphs. Since there are many more hypergraphs than DAGs, undirected graphs, chain graphs, and, indeed, other graph-based networks, as discussed in Remark 13, Bayesian hypergraphs can model much finer factorizations and thus are more computationally efficient. When tied to probability distributions, directed acyclic hypergraphs specify independence (and possibly other) constraints through their Markov properties; we call the new PGM resulting from the directed acyclic hypergraphs and their Markov properties Bayesian hypergraphs. We provide such properties and show that they are consistent with the ones used in Bayesian networks, Markov networks, and LWF chain graphs, when the directed acyclic hypergraphs are suitably restricted. In particular, we define a projection operator, called shadow, that maps a Bayesian hypergraph to a chain graph, and show that the Markov properties of a Bayesian hypergraph are equivalent to those of its shadow (which is a chain graph). This also allows people to work with familiar separation criteria even in the case of Bayesian hypergraphs.

There are situations that may be of interest to a probabilistic or causal modeler that can
be modeled more explicitly using Bayesian hypergraphs. In particular, some causal patterns, such as independence of causal influence (e.g., Noisy-OR), can be expressed graphically in Bayesian hypergraphs, while they require a numerical specification in DAGs or chain graphs. This is one of the important limitations to the Causal Representation Convention (Spirtes, Glymour, and Scheines, 2000). For example, suppose diseases $A$ and $B$ both increase symptoms $C$, but the effect of $B$ without $A$ is quite trivial, while the effect of $A$ alone is not (see Table 5.1(b)). The directed graph representation we have considered in Figure 5.1, which is a DAG, offer no means to represent this interaction and to distinguish it from other circumstances in which $A$ and $B$ alone each have an effect on $C$ (see Table 5.1(a)). Both interactions are only represented through the probability distribution associated with the graph. We describe this case in detail in Section 5.3.2, and we show that how using hypergraphs can help modelers to overcome this problem.

![Figure 5.1: A simple DAG $G$.](image)

Table 5.1: The conditional probability distribution $P(C|A, B)$ for a model with: (a) noisy functional dependence (Noisy-OR), and therefore $P(C = n|A = y, B = y) = P(C = n|A = y)P(C = n|B = y)$; (b) non-noisy functional dependence.

<table>
<thead>
<tr>
<th></th>
<th>A=n</th>
<th>A=y</th>
</tr>
</thead>
<tbody>
<tr>
<td>B=n</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.1</td>
</tr>
<tr>
<td>B=y</td>
<td>0</td>
<td>0.9</td>
</tr>
</tbody>
</table>

We provide a causal interpretation of Bayesian hypergraphs that extends the causal interpretation of LWF chain graphs (Lauritzen and Richardson, 2002), by giving corresponding formulas and a graphical criterion for intervention.
The chapter is organized as follows: In Section 5.1 we introduce some common notations, terminology and concepts on graphs and hypergraphs. In Section 5.2 we review the Markov properties and factorizations in the case of undirected graphs. In Section 5.3, we introduce the Bayesian hypergraphs model, discuss the factorizations, Markov properties and its relations to chain graphs. In Section 5.4, we discuss how interventions can be achieved in Bayesian hypergraphs. Section 7 concludes the chapter and includes some directions for further work.

5.1 Basic Definitions and Concepts

In this chapter, we use \([n]\) to denote the set \(\{1, 2, \ldots, n\}\). For \(a, b \in \mathbb{Z}\), we use \([a, b]\) to denote \(\{k \in \mathbb{Z}: a \leq k \leq b\}\). Given a set \(h\), we use \(|h|\) to denote the number of elements in \(h\).

5.1.1 Graphs

A graph \(G = (V, E)\) is an ordered pair \((V, E)\) where \(V\) is a finite set of vertices (or nodes) and \(E \subseteq V \times V\) consists of a set of ordered pairs of vertices \((v, w) \in V \times V\). Given a graph \(G\), we will use \(V(G), E(G)\) to denote the set of vertices and edges of \(G\) respectively. An edge \((v, w) \in E\) is directed if \((w, v) \notin E\) and undirected if \((w, v) \in E\). We write \(v \rightarrow w\) if \((v, w)\) is directed and \(v \leftarrow w\) if \((v, w)\) is undirected. If \(v \rightarrow w\) then we call \(v\) a neighbor of \(w\) and vice versa. If \(v \rightarrow w\), then we call \(v\) a parent of \(w\) and \(w\) a child of \(v\). Let \(pa_G(v)\) and \(nb_G(v)\) denote the set of parents and neighbors of \(v\), respectively. We say \(v\) and \(w\) are adjacent if either \((v, w) \in E\) or \((w, v) \in E\), i.e., either \(v \rightarrow w\), \(w \rightarrow v\) or \(v \leftarrow w\). We say an edge \(e\) is incident to a vertex \(v\) if \(v\) is contained in \(e\). We also define the boundary \(bd(v)\) of \(v\) by

\[
bd(v) = nb(v) \cup pa(v).
\]

Moreover, given \(\tau \subseteq V\), define

\[
pa_G(\tau) = \left( \bigcup_{v \in \tau} pa_G(v) \right) \setminus \tau.
\]
For every graph $G = (V, E)$, we will denote the underlying undirected graph $G^u = (V, E^u)$, i.e., $E^u = \{(v, u) : (v, u) \in E \text{ or } (u, v) \in E\}$. A path in $G$ is a sequence of distinct vertices $v_0, \ldots, v_k$ such that $(v_i, v_{i+1}) \in E$ for all $0 \leq i \leq k - 1$. A path $v_0, \ldots, v_k$ is directed if for all $0 \leq i < k$, $(v_i, v_{i+1})$ is a directed edge, i.e., $(v_i, v_{i+1}) \in E$ but $(v_{i+1}, v_i) \notin E$. A cycle is a path with the modification that $v_k = v_0$. A cycle is partially directed if at least one of the edges in the cycle is a directed edge. A graph $G$ is acyclic if $G$ contains no partially directed cycle. A vertex $v$ is said to be an anterior of a vertex $u$ if there is a path from $v$ to $u$. We remark that every vertex is also an anterior of itself. If there is a directed path from $v$ to $u$, we call $v$ an ancestor of $u$ and $u$ a descendent of $v$. Moreover, $u$ is a non-descendent of $v$ if $u$ is not a descendent of $v$. Let $\text{ant}(u)$ and $\text{an}(u)$ denote the set of anteriors and ancestors of $u$ in $G$ respectively. Let $\text{de}(v)$ and $\text{nd}(v)$ denote the set of descendents and non-descendents of $v$ in $G$ respectively. For a set of vertices $\tau$, we also define $\text{ant}(\tau) = \{\text{ant}(v) : v \in \tau\}$. Again, note that $\tau \subseteq \text{ant}(\tau)$.

A subgraph of a graph $G$ is a graph $H$ such that $V(H) \subseteq V(G)$ and each edge present in $H$ is also present in $G$ and has the same type. An induced subgraph of $G$ by a subset $A \subseteq V(G)$, denoted by $G_A$ or $G[A]$, is a subgraph of $G$ that contains all and only vertices in $A$ and all edges of $G$ that contain only vertices in $A$. A clique or complete graph with $n$ vertices, denoted by $K_n$, is a graph such that every pair of vertices is connected by an undirected edge.

Now we can define several basic graph representations used in probabilistic graphical models. An undirected graph is a graph such that every edge is undirected. A directed acyclic graph (DAG) is a graph such that every edge is directed and contains no directed cycles. A chain graph is a graph without partially directed cycles. Define two vertices $v$ and
u to be equivalent if there is an undirected path from v to u. Then the equivalence classes under this equivalence relation are the chain components of G. For a vertex set S, define \( E^*(S) \) as the edge set of the complete undirected graph on S. Given a graph G = (V, E) with chain components \( \{\tau : \tau \in D\} \), the moral graph of G, denoted by \( G^m = (V, E^m) \), is a graph such that \( V(G^m) = V(G) \) and \( E^m = E^u \cup \bigcup_{\tau \in D} E^*(bd(\tau)) \), i.e., the underlying undirected graph, where the boundary w.r.t. G of every chain component is made complete. The moral graphs are natural generalizations to chain graphs of the similar concept for DAGs given in (Lauritzen and Spiegelhalter, 1988; Lauritzen et al., 1990).

5.1.2 Hypergraphs

Hypergraphs are generalizations of graphs such that each edge is allowed to contain more than two vertices. Formally, an (undirected) hypergraph is a pair \( \mathcal{H} = (V, \mathcal{E}) \), where \( V = \{v_1, v_2, \ldots, v_n\} \) is the set of vertices (or nodes) and \( \mathcal{E} = \{h_1, h_2, \ldots, h_m\} \) is the set of hyperedges where \( h_i \subseteq V \) for all \( i \in [m] \). If \(|h_i| = k\) for every \( i \in [m] \), then we say \( \mathcal{H} \) is a \( k \)-uniform (undirected) hypergraph. A directed hyperedge or hyperarc \( h \) is an ordered pair, \( h = (X, Y) \), of (possibly empty) subsets of \( V \) where \( X \cap Y = \emptyset \); \( X \) is the called the tail of \( h \) while \( Y \) is the head of \( h \). We write \( X = T(h) \) and \( Y = H(h) \). We say a directed hyperedge \( h \) is fully directed if none of \( H(h) \) and \( T(h) \) are empty. A directed hypergraph is a hypergraph such that all of the hyperedges are directed. A \((s, t)\)-uniform directed hypergraph is a directed hypergraph such that the tail and head of every directed edge have size \( s \) and \( t \) respectively. For example, any DAG is a \((1, 1)\)-uniform hypergraph (but not vice versa). An undirected graph is a \((0, 2)\)-uniform hypergraph. Given a hypergraph \( \mathcal{H} \), we use \( V(\mathcal{H}) \) and \( E(\mathcal{H}) \) to denote the the vertex set and edge set of \( \mathcal{H} \) respectively.

We say two vertices \( u \) and \( v \) are co-head (or co-tail) if there is a directed hyperedge \( h \) such that \( \{u, v\} \subset H(h) \) (or \( \{u, v\} \subset T(h) \) respectively). Given another vertex \( u \neq v \), we say \( u \) is a parent of \( v \), denoted by \( u \rightarrow v \), if there is a directed hyperedge \( h \) such that \( u \in T(h) \) and \( v \in H(h) \). If \( u \) and \( v \) are co-head, then \( u \) is a neighbor of \( v \). If \( u, v \) are neighbors, we
denote them by \( u - v \). Given \( v \in V \), we define parent (\( pa(v) \)), neighbor (\( nb(v) \)), boundary (\( bd(v) \)), ancestor (\( an(v) \)), anterior (\( ant(v) \)), descendant (\( de(v) \)), and non-descendant (\( nd(v) \)) for hypergraphs exactly the same as for graphs (and therefore use the same names). The same holds for the equivalent concepts for \( \tau \subseteq V \). Note that it is possible that some vertex \( u \) is both the parent and neighbor of \( v \).

A partially directed cycle in \( \mathcal{H} \) is a sequence \( \{v_1, v_2, \ldots, v_k\} \) satisfying that \( v_i \) is either a neighbor or a parent of \( v_{i+1} \) for all \( 1 \leq i \leq k \) and \( v_i \rightarrow v_{i+1} \) for some \( 1 \leq i \leq k \). Here \( v_{k+1} \equiv v_1 \). We say a directed hypergraph \( \mathcal{H} \) is acyclic if \( \mathcal{H} \) contains no partially directed cycle. For ease of reference, we call a directed acyclic hypergraph a DAH or a Bayesian hypergraph structure (as defined in Section 5.3). Note that for any two vertices \( u, v \) in a directed acyclic hypergraph \( \mathcal{H} \), \( u \) can not be both the parent and neighbor of \( v \) otherwise we would have a partially directed cycle.

**Remark 6.** DAHs are generalizations of undirected graphs, DAGs and chain graphs. In particular an undirected graph can be viewed as a DAH in which every hyperedge is of the form \((\emptyset, \{u, v\})\). A DAG is a DAH in which every hyperedge is of the form \((\{u\}, \{v\})\). A chain graph is a DAH in which every hyperedge is of the form \((\emptyset, \{u, v\}) \) or \((\{u\}, \{v\})\).

We define the *chain components* of \( \mathcal{H} \) as the equivalence classes under the equivalence relation where two vertices \( v_1, v_t \) are equivalent if there exists a sequence of distinct vertices \( v_1, v_2, \ldots, v_t \) such that \( v_i \) and \( v_{i+1} \) are co-head for all \( i \in [t - 1] \). The chain components \( \{\tau : \tau \in \mathcal{D}\} \) yields an unique natural partition of the vertex set \( V(\mathcal{H}) = \bigcup_{\tau \in \mathcal{D}} \tau \) with the following properties:

**Proposition 5.1.** Let \( \mathcal{H} \) be a DAH and \( \{\tau : \tau \in \mathcal{D}\} \) be its chain components. Let \( G \) be a graph obtained from \( \mathcal{H} \) by contracting each element of \( \{\tau : \tau \in \mathcal{D}\} \) into a single vertex and creating a directed edge from \( \tau_i \in V(G) \) to \( \tau_j \in V(G) \) in \( G \) if and only if there exists a hyperedge \( h \in E(\mathcal{H}) \) such that \( T(h) \cap \tau_i \neq \emptyset \) and \( H(h) \cap \tau_j \neq \emptyset \). Then \( G \) is a DAG.
Proof. First of all, clearly $G$ is a directed graph. Now since $\mathcal{H}$ is a DAH, there is no directed hyperedge such that both its head and tail intersect a common chain component. Hence $G$ has no self-loop. It remains to show that there is no directed cycle in $G$. Suppose for contradiction that there is a directed cycle $\tau_1, \tau_2, \ldots, \tau_k$ in $G$. Then by the construction of $G$, there is a sequence of hyperedges $\{h_1, h_2, \ldots, h_k\}$ such that $T(h_i) \cap \tau_i \neq \emptyset$ and $H(h_i) \cap \tau_{i+1} \neq \emptyset$ (with $\tau_{k+1} \equiv \tau_1$). Since there is a path between any two vertices in the same component, it follows that there is a partially directed cycle in $\mathcal{H}$, which contradicts that $\mathcal{H}$ is acyclic. Hence we can conclude that $G$ is indeed a DAG.

Note that the DAG obtained in Proposition 5.1 is unique and given a DAH $\mathcal{H}$ we call such $G$ the canonical DAG of $\mathcal{H}$. A chain component $\tau$ of $\mathcal{H}$ is terminal if the out degree of $\tau$ in $G$ is 0, i.e., there is no $\tau' \neq \tau$ such that $\tau \rightarrow \tau'$ in $G$. A chain component $\tau$ is initial if the in degree of $\tau$ in $G$ is 0, i.e., there is no $\tau' \neq \tau$ such that $\tau' \rightarrow \tau$ in $G$. We call a vertex set $A \subseteq V(\mathcal{H})$ an anterior set if it can be generated by stepwise removal of terminal chain components. We call $A$ an ancestral set if $\text{bd}(A) = \emptyset$ in $\mathcal{H}$. We remark that given a set $A$, $\text{ant}(A)$ is also the smallest ancestral set containing $A$.

A sub-hypergraph of $\mathcal{H} = (V,E)$ is a directed hypergraph $\mathcal{H}' = (V',E')$ such that $V' \subseteq V$ and $E' \subseteq E$. Given $S \subseteq V(\mathcal{H})$, we say a directed hypergraph $\mathcal{H}'$ is a sub-hypergraph of $\mathcal{H}$ induced by $S$, denoted by $\mathcal{H}_S$ or $\mathcal{H}[S]$, if $V(\mathcal{H}') = S$ and $h \in E(\mathcal{H}')$ if and only if $h \in E(\mathcal{H})$ and $H(h) \cup T(h) \subseteq S$.

To illustrate the relationship between a directed acyclic hypergraph and a chain graph, we will introduce the concept of a shadow of a directed acyclic hypergraph. Given a directed acyclic hypergraph $\mathcal{H}$, let the (directed) shadow of $\mathcal{H}$, denoted by $\partial(\mathcal{H})$, be a graph $G$ such that $V(G) = V(\mathcal{H})$, and for every hyperedge $h = (X,Y) \in E(\mathcal{H})$, $G[Y]$ is a clique (i.e. every two vertices in $G[Y]$ are neighbors) and there is a directed edge from each vertex of $X$ to each vertex of $Y$ in $G$.

**Proposition 5.2.** Suppose $\mathcal{H}$ is a directed acyclic hypergraph and $G$ is the shadow of $\mathcal{H}$. 

172
Then

(i) $G$ is a chain graph.

(ii) For every vertex $v \in V(\mathcal{H}) = V(G)$, \(nb_G(v) = nb_H(v)\) and \(pa_G(v) = pa_H(v)\).

Proof. For (i), note that since $\mathcal{H}$ is acyclic, there is no partially directed cycle in $\mathcal{H}$. It follows by definition that there is no partially directed cycle in $G$. Hence, the shadow of a directed acyclic hypergraph is a chain graph. (ii) is also clear from the definition of the shadow.

Figure 5.2: (1) a DAH $\mathcal{H}$. (2) the canonical DAG of $\mathcal{H}$. (3) the shadow of $\mathcal{H}$.

5.1.3 Hypergraph Drawing

In this subsection, we present how directed edges are drawn in this paper and illustrate the concepts with an example. For a fully directed hyperedge with two vertices (both head and tail contain exactly one vertex), we use the standard arrow notation. For a fully directed hyperedge with at least three vertices, we use a shaded polygon to represent that edge, with the darker side as the head and the lighter side as the tail. For hyperedges of the type \((\emptyset, A)\), we use an undirected line segment (i.e. $-\$) to denote the hyperedge if $|A| = 2$ and a shaded polygon with uniform gray color if $|A| \geq 3$. For example, in Figure 5.2, the directed hyperedges are $((a, b), \{c\}), (\{a\}, \{c, d\}), ([d], \{e, f\}), ([c], \{e\})$. Here $a$ and $b$ are co-tail, $c$ and $d$, $e$ and $f$ are co-head. Figure 5.2 (2) shows the canonical DAG associated to $\mathcal{H}$ with four chain components: $\{a\}, \{b\}, \{c, d\}, \{e, f\}$. Figure 5.2 (3) shows the shadow of $\mathcal{H}$.
5.1.4 Construction of a directed acyclic hypergraph from chain graph

In this subsection, we show how to construct a directed acyclic hypergraph $\mathcal{H}$ from a chain graph $G$ according to the LWF interpretation. We will then show later in Section 5.3.3 that the DAH model $\mathcal{H}$ constructed from $G$ in this section admits the same Markov properties and factorization decomposition as the chain graph model $G$. This shows that the DAH model indeed generalizes the chain graph model (at least according to the LWF interpretation). Due to the expressiveness and generality of a directed hypergraph, other constructions may exist too. Let $G$ be a chain graph with $n$ vertices. We will explicitly construct a directed acyclic hypergraph $\mathcal{H}$ on $n$ vertices that correspond to $G$. We remark that the construction essentially creates a hyperedge for each maximal clique in the moral graph of $G_{c(\tau)}$ for every chain component $\tau$ of $\mathcal{H}$.

Construction:

$$V(\mathcal{H}) = V(G).$$

The edge set of $\mathcal{H}$ is constructed in two phases:

Phase I:

- For each $v \in V(G)$, let $S_v$ be the set of children of $v$ in $G$. Consider the subgraph $G'$ of $G[S_v]$ induced by the undirected edges in $G[S_v]$. For each maximal clique (with vertex set $K$) in $G'$, add the directed hyperedge ($\{v\}, K$) into $\mathcal{H}$.

- Let $\mathcal{H}'$ be the resulting hypergraph after performing the above procedure for every $v \in V(G)$. Now for every maximal clique $K$ (every edge in $K$ is undirected) in $G$, if $K \not\subseteq H(h)$ for every $h \in E(\mathcal{H}')$, add the directed hyperedge $(\emptyset, K)$ into $\mathcal{H}$.

Phase II: Let $\mathcal{H}'$ be the resulting hypergraph constructed from Phase I and $\{\tau : \tau \in \mathcal{D}\}$ be the chain components of $G$. Given $\tau$, let $\mathcal{H}_{\tau}^*$ be the hypergraph containing all the edges $h$ in $\mathcal{H}'_{c(\tau)}$ such that $H(h) \cap \tau \neq \emptyset$. 

174
Define
\[ E(\mathcal{H}) = \bigcup_{\tau \in \mathcal{D}} \left\{ \left( \bigcup_{h \in E(\mathcal{H}_\tau)} T(h), B \right) : B = \bigcap_{h \in F} H(h), \mathcal{F} \subseteq E(\mathcal{H}_\tau) \right\}. \]

Note that the resulting hypergraph \( \mathcal{H} \) is a directed acyclic hypergraph since a partially directed cycle \( C \) in \( \mathcal{H} \) corresponds to a directed cycle in \( G \). Moreover, the above construction gives us an injection from the family of chain graphs with \( n \) vertices to the family of directed acyclic hypergraphs with \( n \) vertices.

Figure 5.3: (1) a simple chain graph \( G \); (2) the corresponding DAH of \( G \) in the LWF interpretation.

Figure 5.3 contains an example of a simple chain graph and its corresponding version in the hypergraph representation. Recall that every fully directed hyperedge is represented (in the drawing) by a colored convex region. The darker side is the head and the lighter side is the tail. We will detail the hyperedges existing in every phase of the construction:

- **Phase I**: the hyperedges in \( \mathcal{H} \) are \( \{a, d, e\} \) and \( \{b, e, f\} \) and \( \{c, f\} \).

- **Phase II**: For each chain component \( \tau \), we obtain all subsets \( B \) of \( \tau \) which are the intersections of the heads of some set of hyperedges intersecting \( \tau \). For each such \( B \) obtained, create a hyperedge whose head is \( B \) and whose tail is the union of the tails of the hyperedges containing \( B \) in its head. In Figure 5.3, for the chain component \( \{d, e, f\} \), the set of such \( B \)'s is \( \{\{d, e\}, \{e, f\}, \{f\}\} \). This is because \( \{d, e\} \) is the head of the hyperedge \( \{a, d, e\} \); \( \{e\} \) is the intersection of the heads of the hyperedges \( \{a, d, e\}, \{a, b, e\} \) and \( \{b, e, f\} \), etc. Hence

\[
E(\mathcal{H}) = \left\{ \{a, d, e\}, \{e, a, b\}, \{b, e, f\}, \{b, c, f\} \right\}.
\]
In this particular example, the hyperedges \{a, b, e\} and \{b, c, f\} are created so that the resulting factorization according to the DAH is consistent with the LWF chain graph model: in particular, the factorization decomposition includes the cliques created due to the moralization operation in the chain graph model.

Hence the resulting hypergraph \(\mathcal{H}\) is the one in Figure 5.3(2).

For ease of reference, given a chain graph, we will call the hypergraph \(\mathcal{H}\) constructed above the *canonical LWF DAH* of \(G\). We say \(\mathcal{H}\) is *hypermoralized* from \(G\) if \(\mathcal{H}\) is the canonical LWF DAH of \(G\). Moreover, we call the family of all such hypergraphs (i.e. the canonical LWF DAH of some chain graph) *LWF DAHs*.

![Figure 5.4: Relationship between chain graphs and directed acyclic hypergraphs](image)

**Remark 7.** In this section, we gave an injective mapping from the space of chain graphs to the space of directed acyclic hypergraphs. The LWF DAH constructed from a LWF chain graph \(G\) has the same Markov properties as \(G\). We show that later in Theorem 5.10. We believe some other types of chain graphs can be modeled by DAHs too (e.g. MVR DAHs) but we do not explore them in this chapter.

**Proposition 5.3.** Let \(G\) be a chain graph and \(\mathcal{H}\) be its canonical LWF DAH. Then we have

(i) For each vertex \(v \in V(G) = V(\mathcal{H})\), \(nb_G(v) = nb_{\mathcal{H}}(v)\) and \(pa_G(v) = pa_{\mathcal{H}}(v)\).

(ii) \(G\) is the shadow of \(\mathcal{H}\).

(iii) \(\mathcal{H}\) is a directed acyclic hypergraph.
Proof. We will first show (i). Note that by our construction in Phase I, if two vertices are neighbors in $G$, then they are co-head in $\mathcal{H}$. Moreover, if $u$ is the parent of $v$ in $G$, then $u$ is still the parent of $v$ in $\mathcal{H}$. These relations remain true in Phase II. Hence we obtain that $\text{nb}_{G}(v) \subseteq \text{nb}_{\mathcal{H}}(v)$ and $\text{pa}_{G}(v) \subseteq \text{pa}_{\mathcal{H}}(v)$ for all $v \in V(\mathcal{H})$. It remains to show that for each $v \in V(\mathcal{H})$, no additional neighbor or parent of $v$ (compared to the case in $G$) is added in the construction. In Phase I, every hyperedge added is either of the form $(\emptyset, K)$ or $([w], K)$ where $K \subseteq V$ induces a complete undirected graph in $G$ and $w$ is the parent of every element in $K$. Hence for every $v \in V(\mathcal{H})$, no additional neighbor or parent of $v$ is added in Phase I. Now let us examine Phase II. Given an edge $h = (A, B) \in E(\mathcal{H})$, there exists some $\tau \in D(G)$ such that $B = \bigcap_{h \in F} H(h)$ for some $F \subseteq E(\mathcal{H}^*_\tau)$. Moreover, $A = \bigcup_{h \in E(\mathcal{H}^*_\tau)} T(h)$. Note for every pair of elements $u, v \in B$, $u, v$ are already neighbors in $G$ since $u, v \in H(h)$ for some $h \in F$ from Phase I. Moreover, for every $v \in A, u \in B$, $v$ is already a parent of $u$ in $G$ since there exists some $h$ constructed in Phase I such that $u \in H(h)$ and $v \in T(h)$. Therefore, it follows that any edge defined in Phase II does not create any new neighbor or parent for any $v \in V(G)$. Thus, we can conclude that for all $v \in V(G) = V(\mathcal{H})$, $\text{nb}_{G}(v) = \text{nb}_{\mathcal{H}}(v)$ and $\text{pa}_{G}(v) = \text{pa}_{\mathcal{H}}(v)$.

(ii) is implied by (i) by the definition of a shadow. (iii) is implied by (ii) since $G$ is acyclic and $G$ is the shadow of $\mathcal{H}$. □

5.2 Markov properties for undirected graphs

In this section, we will summarize some basic results on the Markov properties of undirected graphs. Let us first introduce some notations. In the rest of this week, let $(X_{a})_{a \in V}$ be a collection of random variables taking values in some product space $X = \times_{a \in V} X_{a}$. Let $P$ denote a probability measure on $X$. For a subset $A$ of $V$, we use $X_{A}$ to denote $\times_{a \in A} X_{a}$ and $P_{A}$ is the marginal measure on $X_{A}$. A typical element of $X_{A}$ is denoted by $x_{A} = (x_{a})_{a \in A}$. We will use the short notation $A \perp B \mid C$ for $X_{A} \perp X_{B} \mid X_{C}$.

Given an undirected graph $G$, we say $C$ separates $A$ and $B$ in $G$ if there is no path
from any vertex in $A$ to any vertex in $B$ in $G[V(G)\setminus C]$. If $G$ is an undirected graph, then a probability measure $P$ is said to be:

**(UP) pairwise $G$-Markovian** if $\alpha \perp \beta \mid V\setminus\{\alpha,\beta\}$ whenever $\alpha$ and $\beta$ are non-adjacent in $G$.

**(UL) local $G$-Markovian** if $\alpha \perp V\setminus cl(\alpha) \mid bd(\alpha)$ for all $\alpha \in V(G)$.

**(UG) global $G$-Markovian** if $A \perp B \mid C$ whenever $C$ separates $A$ and $B$ in $G$.

The following theorem by Pearl and Paz (Pearl and Paz, 1987) gives a sufficient condition for the equivalence of (UG), (UL) and (UP):

**Theorem 5.4.** If $G$ is an undirected graph and $P$ satisfies the intersection property ($S5$), then (UG), (UL) and (UP) are equivalent and $P$ is said to be $G$-Markovian if they hold.

Conditional independences and thus Markov properties are closely related to factorizations. A probability measure $P$ on $X$ is said to **factorize** according to $G$ if for each clique $h$ in $G$, there exist a non-negative function $\psi_h$ depending on $x_h$ only and there exists a product measure $\mu = \times_{\alpha \in V} \mu_\alpha$ on $X$ such that $P$ has density $f$ with respect to $\mu$ where $f$ has the form

$$f(x) = \prod_{h \in C} \psi_h(x)$$

where $C$ is the set of maximal cliques in $G$. If $P$ factorizes according to $G$, we say $P$ has property (UF). It is known (see Lauritzen (1996)) that

$$(UF) \implies (UG) \implies (UL) \implies (UP).$$

Moreover, in the case that $P$ has a positive and continuous density, it can be proven using Möbius inversion lemma that $(UP) \implies (UF)$. This result seems to have been discovered in various forms by a number of authors and is usually attributed to Hammersley and Clifford who proved the result in the discrete case (Lauritzen, 1996).
5.3 **Bayesian Hypergraphs**

A *Bayesian hypergraph* (BH) is a probabilistic graphical model that represents a set of variables and their conditional dependencies through an acyclic directed hypergraph $\mathcal{H}$. Hypergraphs contain many more edges than chain graphs. Thus a Bayesian hypergraph is a more general and powerful framework for studying conditional independence relations that arise in various statistical contexts.

5.3.1 **Markov Properties of Bayesian Hypergraphs**

Analogous to chain graph’s case, we can define the Markov properties of a Bayesian hypergraph in a variety of ways. Let $\mathcal{H}$ be a directed acyclic hypergraph with chain components $\{\tau : \tau \in D\}$. We say that a probability measure $P$ defined on $X = \times_{\alpha \in V(\mathcal{H})} X_\alpha$ is:

(HP) *pairwise $\mathcal{H}$-Markovian*, relative to $\mathcal{H}$, if for every pair $(v, u)$ of non-adjacent vertices in $\mathcal{H}$ with $u \in nd(v)$,

$$v \perp u \mid nd(v) \backslash \{v, u\}. \tag{5.2}$$

(HL) *local $\mathcal{H}$-Markovian*, relative to $\mathcal{H}$, if for any vertex $v \in V(\mathcal{H})$,

$$v \perp nd(v) \backslash cl(v) \mid bd(v). \tag{5.3}$$

(HG) *global $\mathcal{H}$-Markovian*, relative to $\mathcal{H}$, if for all $A, B, C \subseteq V$ such that $C$ separates $A$ and $B$ in $(\partial(\mathcal{H}_{\text{ant}}(A \cup B \cup C)))^m$, the moral graph of the (directed) shadow of the smallest ancestral set containing $A \cup B \cup C$, we have

$$A \perp B \mid C.$$

**Definition 5.5.** A *Bayesian hypergraph* is a triple $(V, \mathcal{H}, P)$ such that $V$ is a set of random variables, $\mathcal{H}$ is a DAH on the vertex set $V$ and $P$ is a multivariate probability distribution on $V$ such that the local Markov property, i.e., (HL), holds with respect to the DAH $\mathcal{H}$. 

179
Given a Bayesian hypergraph \((V, \mathcal{H}, P)\), we call \(\mathcal{H}\) the \textit{Bayesian hypergraph structure}\footnote{or \textit{U-structure} (Cox and Wermuth, 1996)} or the \textit{underlying DAH} of the Bayesian hypergraph. For ease of reference, we simply use \(\mathcal{H}\) to denote the Bayesian hypergraph. Moreover, for a Bayesian hypergraph \(\mathcal{H}\) whose underlying DAH is a LWF DAH, we call \(\mathcal{H}\) a \textit{LWF Bayesian hypergraph}.

**Remark 8.** Observe that by Proposition 5.2 and the definitions of the hypergraph Markov properties, a Bayesian hypergraph has the same pairwise, local and global Markov properties as its shadow, which is a chain graph. Consequently, any separation criteria (e.g., \(d\)-separation) for LWF chain graphs also apply to the shadow of the LWF Bayesian hypergraphs. This allows people to work with familiar separation criteria even in the case of Bayesian hypergraphs.

By Remark 8, we can derive the following corollaries from results on the Markov properties of chain graphs:

**Corollary 5.6.**

\[ (HG) \implies (HL) \implies (HP). \]

Furthermore, if we assume (2.2), then the global, local and pairwise Markov properties are equivalent.

**Corollary 5.7.** Assume that \(P\) is such that (2.2) holds for disjoint subsets of \(V\). Then

\[ (HG) \iff (HL) \iff (HP). \]

**Proof.** This follows from Remark 8 and Theorem 2.1. \(\square\)

Given a chain graph \(G\), a triple \((\alpha, B, \beta)\) is a \textit{complex} in \(G\) if \(B\) is a connected subset of a chain component \(\tau\), and \(\alpha, \beta\) are two non-adjacent vertices in \(bd(\tau) \cap bd(B)\). Moreover, \((\alpha, B, \beta)\) is a \textit{minimal complex} if \(B = B'\) whenever \(B'\) is a subset of \(B\) and \((\alpha, B', \beta)\) is a complex. Frydenberg (Frydenberg, 1990) showed that two chain graphs have the same
Markov properties if they have the same underlying undirected graph and the same minimal complexes. In the case of a Bayesian hypergraph, by Remark 8 and the result on the Markov equivalence of chain graphs, we obtain the following conclusion on the Markov equivalence of Bayesian hypergraphs.

**Corollary 5.8.** Two Bayesian hypergraphs have the same Markov properties if their shadows are Markov equivalent, i.e., their shadows have the same underlying undirected graph and the same minimal complexes.

### 5.3.2 Factorization according to Bayesian hypergraphs

The factorization of a probability measure $P$ according to a Bayesian hypergraph is similar to that of a chain graph. Before we present the factorization property, let us introduce some additional terminology.

Given a DAH $\mathcal{H}$, we use $\mathcal{H}^u$ to denote the undirected hypergraph obtained from $\mathcal{H}$ by replacing each directed hyperedge $h = (A, B)$ of $\mathcal{H}$ into an undirected hyperedge $A \cup B$. Given a family of sets $\mathcal{F}$, define a partial order $(\mathcal{F}, \leq)$ on $\mathcal{F}$ such that for two sets $A, B \in \mathcal{F}$, $A \leq B$ if and only if $A \subseteq B$. Let $\mathcal{M}(\mathcal{F})$ denote the set of maximal elements in $\mathcal{F}$, i.e., no element in $\mathcal{M}(\mathcal{F})$ contains another element as subset. When $\mathcal{F}$ is a set of directed hyperedges, we abuse the notation to denote $\mathcal{M}(\mathcal{F}) = \mathcal{M}(\mathcal{F}^u)$.

Let $\mathcal{H}$ be a directed acyclic hypergraph, $\mathcal{D}$ be the canonical DAG of the chain components of $\mathcal{H}$. Assume that a probability distribution $P$ has a density $f$, with respect to some product measure $\mu = \times_{a \in V} \mu_a$ on $X = \times_{a \in V} X_a$. Now we say a probability measure $P$ factorizes according to $\mathcal{H}$ if it has density $f$ such that

(i) $f$ factorizes as in the directed acyclic case:

$$f(x) = \prod_{\tau \in \mathcal{D}} f(x_\tau \mid x_{\text{pa}(\tau)}).$$

(ii) For each $\tau \in \mathcal{D}$, define $\mathcal{H}_\tau'$ to be the subhypergraph of $\mathcal{H}_{\tau \cup \text{pa}(\tau)}$ containing all edges $h$ in $\mathcal{H}_{\tau \cup \text{pa}(\tau)}$ such that $H(h) \subseteq \tau$. 

181
\[ f(x_{\tau} \mid x_{\text{pa}(\tau)}) = \prod_{h \in M(\mathcal{H}_{\tau})} \psi_h(x). \] (5.5)

where \( \psi_h \) are non-negative functions depending only on \( x_h \) and

\[ \int_{X_{\tau}} \prod_{h \in M(\mathcal{H}_{\tau})} \psi_h(x) \mu_{\tau}(dx_{\tau}) = 1. \]

Equivalently, we can also write \( f(x_{\tau} \mid x_{\text{pa}(\tau)}) \) as

\[ f(x_{\tau} \mid x_{\text{pa}(\tau)}) = Z^{-1}(x_{\text{pa}(\tau)}) \prod_{h \in M(\mathcal{H}_{\tau})} \psi_h(x), \] (5.6)

where

\[ Z^{-1}(x_{\text{pa}(\tau)}) = \int_{X_{\tau}} \prod_{h \in M(\mathcal{H}_{\tau})} \psi_h(x) \mu_{\tau}(dx_{\tau}). \]

**Remark 9.** Note that although (LWF) Bayesian hypergraphs are generalizations of Bayesian networks and LWF chain graph models, the underlying graph structures that represent the same factorizations may differ. The motivation behind this representational choice is that it is easier for a modeler to represent induced dependencies explicitly via directed hyperedges, rather than representing the absence of induced dependencies via directed hyperedges. As a consequence, the underlying graph structures of Bayesian networks and chain graph do not directly migrate to Bayesian hypergraphs.

![Figure 5.5: A simple Bayesian hypergraph \( \mathcal{H} \).](image)

We will illustrate with an example. Consider the graph in Figure 5.5, which can be interpreted as a chain graph structure \( G \) or a Bayesian hypergraph structure \( \mathcal{H} \). Note that the factorizations, under the two interpretations, are different. In particular, the factorization, according to \( G \), is

\[ f_G(x) = f(x_a)f(x_b)\psi_{abc}(x) \]
for some non-negative functions $\psi_{abc}$. On the other hand, the factorization, according to $H$, is

$$f_H(x) = f(x_a)f(x_b)\psi_{ac}(x)\psi_{bc}(x)$$

for some non-negative functions $\psi_{ac}, \psi_{bc}$.

![Figure 5.6: (1) a chain graph $G$; (2) a Bayesian hypergraph $H$.](image)

**Remark 10.** One of the key advantages of Bayesian hypergraphs is that they allow much finer factorizations of probability distributions compared to chain graph models. We will illustrate with a simple example in Figure 5.6. Note that in Figure 5.6 (1), the factorization according to $G$ is

$$f(x) = f(x_a)f(x_b)f(x_{cd} | x_{ab})$$

$$= f(x_a)f(x_b)\psi_{abcd}(x)$$

In Figure 5.6 (2), the factorization according to $H$ is

$$f(x) = f(x_a)f(x_b)f(x_{cd} | x_{ab})$$

$$= f(x_a)f(x_b)\psi_{abc}(x)\psi_{abd}(x)\psi_{cd}(x)$$

Note that although $G$ and $H$ have the same global Markov properties, the factorization according to $H$ is one step further compared to the factorization according to $G$. Suppose each of the variables of $\{a, b, c, d\}$ can take $k$ values. Then the factorization according to $G$ will require a conditional probability table of size $k^4$ while the factorization according to $H$ only needs a table of size $\Theta(k^3)$ asymptotically. Hence, a Bayesian hypergraph model allows much finer factorizations and thus achieves higher memory efficiency.
Remark 11. We remark that the factorization formula defined in (5.5) is in fact the most general possible in the sense that it allows all possible factorizations of a probability distribution admitted by a DAH. In particular, given a Bayesian hypergraph $\mathcal{H}$ and one of its chain components $\tau$, the factorization scheme in (5.5) allows a distinct function for each maximal subset of $\tau \cup \text{pa}_D(\tau)$ that intersects $\tau$ ($\text{pa}_D$ is the parent of $\tau$ in the canonical DAG of $\mathcal{H}$). For each subset $S$ of $\tau \cup \text{pa}_D(\tau)$ that does not intersect $\tau$, recall that the factorization in (5.5) can be rewritten as follows:

$$f(x_\tau \mid x_{\text{pa}(\tau)}) = \left( \prod_{h \in M(\mathcal{H}_\tau^* \tau)} \psi_h(x) \right) / \left( \int_{X_\tau} \prod_{h \in M(\mathcal{H}_\tau^* \tau)} \psi_h(x) \mu_\tau(dx_\tau) \right).$$

Observe that $\psi_S(x)$ is a function that does not depend on values of variables in $\tau$. Hence $\psi_S(x)$ can be factored out from the integral above and cancels out with itself in $f(x_\tau \mid x_{\text{pa}(\tau)})$.

Thus, the factorization formula in (5.5) or (5.6) in fact allows distinct functions for all possible maximal subsets of $\tau \cup \text{pa}_D(\tau)$.

Table 5.2 lists some factorizations of three random variables and the corresponding BH representation. Entry 1 (top left) corresponds to a three-node Bayesian network: an uncoupled converging connection (unshielded collider) at $c$. Entry 3 (below entry 1) corresponds to a three-node Bayesian network like the one in entry 1, with the constraint that the conditional probability table factorizes as, for example, in a Noisy-OR functional dependence and, more generally, in a situation for which compositionality holds, such as MIN, MAX, or probabilistic sum (Pearl, 1993; Hájek, Havránek, and Jirousek, 1992; Jensen and Nielsen, 2007). Graphical modeling languages should capture assumptions graphically in a transparent and explicit way, as opposed to hiding them in tables or functions. By this criterion, the Bayesian hypergraph of entry 3 shows the increased power of our new PGM with respect to Bayesian networks and chain graphs.

For a detailed example of Noisy-OR functional dependence, consider the (much simplified) heart disease model of (Ghosh and Valtorta, 2000), shown in Figure 5.7, and the family of nodes Obesity (O, with values Yes, No), Diet (D, with values Bad, Good), and
Table 5.2: Factorizations and corresponding BH representations

<table>
<thead>
<tr>
<th>Factorization</th>
<th>BH representation</th>
<th>Factorization</th>
<th>BH representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(x) = f(x_a)f(x_b)\psi_{abc}(x)$</td>
<td><img src="image1" alt="Diagram" /></td>
<td>$f(x) = f(x_{ab})\psi_{abc}(x)$</td>
<td><img src="image2" alt="Diagram" /></td>
</tr>
<tr>
<td>$f(x) = f(x_a)f(x_b)\psi_{ac}(x)\psi_{bc}(x)$</td>
<td><img src="image3" alt="Diagram" /></td>
<td>$f(x) = f(x_{ab})\psi_{ac}(x)\psi_{bc}(x)$</td>
<td><img src="image4" alt="Diagram" /></td>
</tr>
<tr>
<td>$f(x) = f(x_a)f(x_b)$</td>
<td><img src="image5" alt="Diagram" /></td>
<td>$f(x) = f(x_{ab})\psi_{ac}(x)\psi_{bc}(x)$</td>
<td><img src="image6" alt="Diagram" /></td>
</tr>
<tr>
<td>$f(x) = \psi_{ac}(x)\psi_{bc}(x)$</td>
<td><img src="image7" alt="Diagram" /></td>
<td>$f(x) = f(x_c)f(x_{ab})\psi_{ac}(x)\psi_{bc}(x)$</td>
<td><img src="image8" alt="Diagram" /></td>
</tr>
<tr>
<td>$f(x) = f(x_c)\psi_{ac}(x)\psi_{bc}(x)$</td>
<td><img src="image9" alt="Diagram" /></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Moderate Exercise (M, with values Yes, No). The Noisy-OR model is used to compute the conditional probability of O given M and D. Good diet prevents obesity, except when an inhibiting mechanism prevents that with probability $q_{D\rightarrow O}$; moderate exercise prevents obesity except when an inhibiting mechanism prevents that with probability $q_{M\rightarrow O}$. The inhibiting mechanisms are independent, and therefore the probability of being obese given both a good diet and moderate exercise is $1 - q_{D\rightarrow O}q_{M\rightarrow O}$. Equivalently, the probability of not being obese given both a good diet and moderate exercise is $q_{D\rightarrow O}q_{M\rightarrow O}$. If we consider a situation with only the variables just described, the joint probability of Diet, Moderate Exercise, and Obesity factorizes exactly as in the Bayesian hypergraph of entry 3, with the caution that only half of the entries in the joint probability table are computed directly; the others are computed by the complement to one.

Similarly, entry 2 corresponds to a three node chain graph, while entry 4 may be used to model a situation in which variables $a$ and $b$ are related by being effects of a common
latent cause, while the mechanisms by which they, in turn, affect variable $c$ are causally independent. While such a situation may be unusual, it is notable that it can be represented graphically in Bayesian hypergraphs. Therefore, the Bayesian hypergraph of entry 4 shows the increased power of our new PGM with respect to Bayesian networks and chain graphs.

For a detailed example, consider again the model shown in Figure 5.7 and, this time, the structure in which Moderate Exercise, Serum LDL (S-LDL), Serum Triglicerides (S-T), and Cholesterol HDL (C-HDL) Ratio are parents (possible causes) of Atheriosclerosis, and Diet is a parent of S-LDL, S-T, and C-HDL. As in the previous example, the Noisy-
OR assumption is made, and therefore, after marginalization of Diet, the computation of the joint probability of Moderate Exercise, S-LDL, S-T, C-HDL, and Atheriosclerosis factorizes as in an entry 4, with a slight generalization due to the presence of four parents instead of two. As in entry 4, the parents (causes) are not marginally independent, due to their common dependence on Diet, but the conditional probability of the effect decomposes multiplicatively.

Moreover, as illustrated in Remark 10 and Table 5.2, a Bayesian hypergraph enables much finer factorization than a chain graph. In the factorization w.r.t. a chain graph $G$ with chain components $\{\tau : \tau \in \mathcal{D}\}$, $f(x_\tau \mid x_{pa(\tau)})$ is only allowed to be further factorized based on the maximal cliques in the moral graph of $G_{\tau \cup pa(\tau)}$, which is rather restrictive. In comparison, a Bayesian hypergraph $\mathcal{H}$ allows factorization based on the maximal elements in all subsets of the power set of $\tau \cup pa_\mathcal{D}(x)$. Finer factorizations have the advantage of memory saving in terms of the size of the probability table required. Moreover, factorizations according to Bayesian hypergraphs can be obtained directly from reading off the hyperedges instead of having to search for all maximal cliques in the moral graph (in the chain graph’s case). Hence, Bayesian hypergraphs enjoy an advantage in heuristic adequacy as well as representational adequacy.

Next, we investigate the relationship between the factorization property and the Markov properties of Bayesian hypergraphs.

**Proposition 5.9.** Let $P$ be a probability measure with density $f$ that factorizes according
to a DAH $\mathcal{H}$. Then

$$(\text{HF}) \implies (\text{HG}) \implies (\text{HL}) \implies (\text{HP}).$$

**Proof.** It suffices to show $(\text{HF}) \implies (\text{HG})$ since the other implications are proven in Corollary 5.6. Let $A, B, C \subseteq V(\mathcal{H})$ such that $C$ separates $A$ and $B$ in $G = (\partial(\mathcal{H}_{\text{ant}(A \cup B \cup C)}))^m$. Let $\tilde{A}$ be the connectivity components in $G \setminus C$ containing $A$ and let $\tilde{B} = V \setminus (\tilde{A} \cup C)$. Note that in $(\partial(\mathcal{H}_{\text{ant}(A \cup B \cup C)}))^m$, every hyperedge $h = (T, H)$ becomes a complete graph on the vertex set $T \cup H$ because of moralization. Observe that since $C$ separates $A$ and $B$ in $G$, for every hyperedge $h = (T, H)$, $T \cup H$ is either a subset of $\tilde{A} \cup C$ or $\tilde{B} \cup C$. Let $\mathcal{H}' = \mathcal{H}_{\text{ant}(A \cup B \cup C)}$ and $\{\tau : \tau \in \mathcal{D}'\}$ be the chain components of $\mathcal{H}'$. For each $\tau \in \mathcal{D}'$, define $\mathcal{H}'_\tau$ to be the subhypergraph of $\mathcal{H}'_{\cup \text{pa}(\tau)}$ containing all edges $h$ in $\mathcal{H}'_{\cup \text{pa}(\tau)}$ such that $H(h) \subseteq \tau$. We then obtain from the (HF) property that

$$f_{\mathcal{H}'}(x) = \prod_{\tau \in \mathcal{D}'} \prod_{h \in M(\mathcal{H}'_\tau)} \psi_h(x) = \phi_1(x_{\tilde{A} \cup C})\phi_2(x_{\tilde{B} \cup C}).$$

for some non-negative functions $\phi_1, \phi_2$. By integrating over the chain components not in $\text{ant}(A \cup B \cup C)$, it follows that

$$f(x) = \psi_1(x_{\tilde{A} \cup C})\psi_2(x_{\tilde{B} \cup C}).$$

for some non-negative functions $\psi_1, \psi_2$. Hence, we have that

$$\tilde{A} \indep \tilde{B} | C.$$

By (S2: Decomposition) property of conditional independences, it follows that $A \indep B | C$. □

**Remark 12.** Due to the generality of factorizations according to Bayesian hypergraphs, the reverse direction of the implication $(\text{HF}) \implies (\text{HG})$ in Proposition 5.9 is generally not true. We will illustrate with the following example.
Consider the two Bayesian hypergraphs $\mathcal{H}_1$ and $\mathcal{H}_2$ in Figure 5.10. Note that they have the same global Markov properties since the shadows of $\mathcal{H}_1$ and $\mathcal{H}_2$ are the same. However the factorizations according to $\mathcal{H}_1$ and $\mathcal{H}_2$ are different. If we let $f_1, f_2$ denote the factorizations represented by $\mathcal{H}_1$ and $\mathcal{H}_2$, then

$$f_1(x) = f_1(x_a)f_1(x_b)f_1(x_{cd} \mid x_{ab})$$
$$= f_1(x_a)f_1(x_b)\psi_{abcd}(x)$$

while

$$f_2(x) = f_2(x_a)f_2(x_b)f_2(x_{cd} \mid x_{ab})$$
$$= f_2(x_a)f_2(x_b)\psi_{abc}(x)\psi_{abd}(x)\psi_{cd}(x)$$

This shows that (HG) does not generally imply (HF).

**Remark 13.** We remark that in our model, two Bayesian hypergraphs that are Markov equivalent may not be factorization-equivalent. This also implies that in general (HF) does not imply (HG). Below we present a combinatorial argument for why this cannot be resolved. We claim that the number of possible forms of factorizations admitted by Bayesian hypergraphs is much more than the number of conditional independence statements over the same set of variables. First, observe that the number of conditional independence statements on $n$ variables is upper bounded by the number of ways to partition $n$ elements into four disjoint sets $A, B, C, D$. Each such partition induces a conditional statement $A \perp B \mid C$ and $D$ is the set of unused variables. There are $4^n$ ways to partition $n$ distinct elements into
four ordered pairwise disjoint sets. Hence there are at most $4^n$ conditional independence statements on $n$ variables.

On the other hand, we give a simple lower bound on the number of directed acyclic hypergraphs by simply counting the number of directed acyclic hypergraphs $\mathcal{H}$ whose vertex sets can be partitioned into two sets $A, B$ such that $|A| = |B| = n/2$ and every fully directed edge has its tail only from $A$ and its head only from $B$. Observe that there are $2^{n/2}$ subsets of $A$ and $B$ respectively. By Sperner’s theorem (Sperner, 1928), the largest number of subsets of $A$ none of which contain any other is upper bounded by $\binom{n/2}{n/4}$. The same holds for $B$. Hence there are at least $\left(\binom{n/2}{n/4}\right)^2$ possible directed hyperedges such that when viewed as undirected hyperedge, no edge contains any other as subset. Therefore, there are at least

$$2^{\left(\binom{n/2}{n/4}\right)^2} = \Theta\left(2^{\frac{2^{n/2}}{\pi}}\right)$$

distinct factorizations admitted by DAHs whose directed edges have their tails only from $A$ and their heads only from $B$. Note that this number is much less than the total number of distinct factorizations admitted by DAHs, but is already much bigger than $4^n$, which is the upper bound on the number of conditional independence statements on $n$ variables. Hence, there are many more factorizations allowed by Bayesian hypergraphs than the number of conditional independence statements on $n$ variables, which suggest that (HG) does not imply (HF) in general.

5.3.3 Comparison between LWF chain graph and LWF Bayesian hypergraph

Let $G$ be a LWF chain graph and $\mathcal{H}$ be the canonical LWF DAH constructed from $G$ described in Section 5.1.4. In this section, we show that $\mathcal{H}$ and $G$ admit the same Markov properties and factorization decomposition. Hence Bayesian hypergraphs generalize the LWF chain graphs.

**Theorem 5.10.** Let $G$ be a chain graph and $\mathcal{H}$ be its canonical (LWF) DAH. We show that a probability measure $P$ satisfies the following:
(i) \( P \) is pairwise G-Markovian if and only if \( P \) is pairwise \( \mathcal{H} \)-Markovian.

(ii) \( P \) is local G-Markovian if and only if \( P \) is local \( \mathcal{H} \)-Markovian.

(iii) \( P \) is global G-Markovian if and only if \( P \) is global \( \mathcal{H} \)-Markovian.

Proof. By Proposition 5.3, \( \text{nb}_G(v) = \text{nb}_\mathcal{H}(v) \), \( \text{pa}_G(v) = \text{pa}_\mathcal{H}(v) \). Hence the same equality holds for \( \text{nd}_G(v), \text{bd}_G(v), \text{cl}_G(v) \), which gives us i and ii by definition of the Markov properties. iii results from the fact that for all \( A, B, C \subseteq V(G) = V(\mathcal{H}) \), \( G_{\text{ant}}(A \cup B \cup C) = \partial(\mathcal{H}_{\text{ant}}(A \cup B \cup C)) \).

\[ \square \]

Theorem 5.11. Let \( G \) be a chain graph and \( \mathcal{H} \) be its canonical LWF DAH. Then a probability measure \( P \) with density \( f \) factorizes according to \( G \) if and only if \( f \) factorizes according to \( \mathcal{H} \).

Proof. Note that by Proposition 5.3, \( G \) and \( \mathcal{H} \) have the same set of chain components \( \{\tau : \tau \in \mathcal{D}\} \). It suffices to show for every \( \tau \in \mathcal{D} \), there exists a bijective map \( \phi \) from the set of maximal edges in \( (\mathcal{H}_\tau)^u \) to the set of maximal cliques in \( (G_{\tau \cup \text{pa}(\tau)})^m \) such that for each maximal edge \( h \) in \( (\mathcal{H}_\tau)^u \), \( \phi(h) = h \). For ease of reference, let \( \mathcal{H}' = (\mathcal{H}_\tau)^u \) and let \( G' = (G_{\tau \cup \text{pa}(\tau)})^m \). Define \( \phi(h) = h \). We need to show two things: (1) for every maximal edge \( h \) in \( \mathcal{H}' \), \( h \) induces a maximal clique in \( G' \); (2) for every maximal clique \( h \) in \( G' \), \( h \) is a maximal edge in \( \mathcal{H}' \).

We first show (1). Suppose that \( h \) is a maximal edge in \( \mathcal{H}' \). Clearly, \( h \) induces a clique in \( G' \) because of the moralization. Suppose for the sake of contradiction that \( h \) is not maximal in \( G' \), i.e. there is a maximal clique \( h' \) in \( G' \) such that \( h \nsubseteq h' \). Let \( h' = A \cup B \) where \( A \subseteq \text{pa}(\tau) \) and \( B \subseteq \tau \). There are two cases:

Case 1: \( A = \emptyset \) or \( B = \emptyset \).

Note that \( B \) cannot be an empty set since \( h \) is an edge in \( \mathcal{H}' \) and every edge in \( \mathcal{H}' = (\mathcal{H}_\tau)^u \) intersects \( \tau \) by definition. If \( A = \emptyset \), then \( h' \) is a maximal clique in \( \tau \). By Phase I of the construction, \( h' \) either is a hyperedge in \( \mathcal{H}' \) or is contained in the head.
of a hyperedge. In either case, since \( h \subseteq h' \), it contradicts that \( h \) is a maximal edge in \( \mathcal{H'} \).

Case 2: \( A \neq \emptyset \) and \( B \neq \emptyset \).

Since \( A \cup B \) induces a maximal clique in \( G' \), it follows that for every \( a \in A, b \in B, a \in pa(b) \). Hence \( B \) the common children of some elements in \( A \). Recall that in Phase I of our construction, for every \( v, (\{v\}, K_v) \) is an hyperedge in \( \mathcal{H} \) where \( K_v \) is a maximal clique in the children of \( v \) in \( G \). Hence \( \mathcal{F} \subseteq E(\mathcal{H}_v^\tau) \) such that \( B \subseteq \cap_{h \in F} \mathcal{H}(h) \). By maximality of \( h' \), \( B = \cap_{h \in F} \mathcal{H}(h) \). Now by our construction in Phase II, there exists a hyperedge

\[
h'' = \left( \bigcup_{h \in E(\mathcal{H}_v^\tau)} T(h), B \right) \in E(\mathcal{H}_v^\tau).
\]

Since every element in \( A \) is a parent of every element in \( B \), it follows that

\[
A \subseteq \bigcup_{h \in E(\mathcal{H}_v^\tau)} T(h).
\]

By maximality of \( A \), it follows that

\[
h \subsetneq h' = h'' \in E(\mathcal{H}_v^\tau).
\]

which contradicts the maximality of \( h \) again.

Hence in both cases, we obtain by contradiction that \( h \) induces a maximal clique in \( G' \).

It remains to show (2). Suppose \( h \) induces a maximal clique in \( G' \). Observe that every hyperedge in \( \mathcal{H'} \) induces a clique in \( G' \). Similar logic and case analysis above apply and it is not hard to see that \( h \) is a maximal edge in \( \mathcal{H'} \). We will leave the details to the reader. \( \Box \)

In Figure 5.11, both \( G \) and its canonical LWF DAH \( \mathcal{H} \) have chain components

\[
\{\{a\}, \{b\}, \{c\}, \{d, e, f\}\}.
\]
Example 21.

Figure 5.11: (1) a simple chain graph $G$; (2) The moral graph $G^m$ of $G$; (3) $\mathcal{M}(\mathcal{H})$ where $\mathcal{H}$ is the canonical LWF DAH of $G$.

Figure 5.11 (2) shows the moral graph $G^m$ of $G$. The maximal cliques in $G^m$ are $\{ade, abce, cef\}$. Thus, by the factorization property of LWF chain graphs, we have that a probability measure $P$ with density $f$ that factorizes according to $G$ satisfies

$$f(x) = f(x_a)f(x_b)f(x_c)f(x_{d,e,f} \mid x_{a,b,c})$$

$$= f(x_a)f(x_b)f(x_c)\psi_{ade}(x)\psi_{abce}(x)\psi_{cef}(x).$$

Figure 5.11 (3) gives the undirected hypergraph with edge set $M(\mathcal{H})$. Observe that $M(\mathcal{H})$ has the same members as the set of maximal cliques in $G^m$. Hence by the factorization property of Bayesian hypergraphs, they admit the same factorization.

5.4 Intervention in Bayesian hypergraphs

Formally, intervention in Bayesian hypergraphs can be defined analogously to intervention in LWF chain graphs (Lauritzen and Richardson, 2002). In this section, we give graphical procedures that are consistent with the intervention formulas for chain graphs (Equation (5.7), (5.8)) and for Bayesian hypergraphs (Equation (5.9), (5.10)). Before we present the details, we need some additional definitions and tools to determine when factorizations according to two chain graphs or DAHs are equivalent in the sense that they could be written as products of the same type of functions (functions that depend on same set of variables).

We say two chain graphs $G_1, G_2$ admit the same factorization decomposition if for every probability density $f$ that factorizes according to $G_1$, $f$ also factorizes according to $G_2$, and vice versa. Similarly, two DAHs $\mathcal{H}_1, \mathcal{H}_2$ admit the same factorization decomposition if for
every probability density \( f \) that factorizes according to \( \mathcal{H}_1 \), \( f \) also factorizes according to \( \mathcal{H}_2 \), and vice versa.

5.4.1 FACTORIZATION EQUIVALENCE AND INTERVENTION IN CHAIN GRAPHS

In this subsection, we will give graphical procedures to model intervention based on the formula introduced by Lauritzen and Richardson in (Lauritzen and Richardson, 2002). Let us first give some background. In many statistical context, we would like to modify the distribution of a variable \( Y \) by intervening externally and forcing the value of another variable \( X \) to be \( x \). This is commonly refered as conditioning by intervention or conditioning by action and denoted by \( \Pr(y \| x) \) or \( \Pr(y \mid X \leftarrow x) \). Other expressions such as \( \Pr(Y_x = y) \), \( P_{\text{man}}(x) \), \( \text{set}(X = x) \), \( X = \hat{x} \) or \( \text{do}(X = x) \) have also been used to denote intervention conditioning (Splawa-Neyman, Dabrowska, and Speed, 1990; Rubin, 1974; Spirtes, Glymour, and Scheines, 2000; Pearl, 1993, 1995, 2009).

Let \( G \) be a chain graph with chain components \( \{ \tau : \tau \in \mathcal{D} \} \). Moreover, assume further that a subset \( A \) of variables in \( V(G) \) are set such that for every \( a \in A \), \( x_a = a_0 \). Lauritzen and Richardson, in (Lauritzen and Richardson, 2002), generalized the conditioning by intervention formula for DAGs and gave the following formula for intervention in chain graphs (where it is understood that the probability of any configuration of variables inconsistent with the intervention is zero). A probability density \( f \) factorizes according to \( G \) (with \( A \) intervened) if

\[
f(x \| x_A) = \prod_{\tau \in \mathcal{D}} f(x_{\tau \setminus A} \mid x_{\text{pa}(\tau)}, x_{\tau \cap A}). \tag{5.7}
\]

Moreover, for each \( \tau \in \mathcal{D} \),

\[
f(x_{\tau \setminus A} \mid x_{\text{pa}(\tau)}, x_{\tau \cap A}) = Z^{-1}(x_{\text{pa}(\tau)}, x_{\tau \cap A}) \prod_{h \in \mathcal{C}} \psi_h(x) \tag{5.8}
\]

where \( \mathcal{C} \) is the set of maximal cliques in \( (G_{\tau \cup \text{pa}(\tau)})^m \) and \( Z^{-1}(x_{\text{pa}(\tau)}, x_{\tau \cap A}) = \int_{x_{\tau \setminus A}} \prod_{h \in \mathcal{C}} \psi_h(x) \mu_{\tau \setminus A}(dx_{\tau \setminus A}) \).
Definition 5.12. Let $G_1$ and $G_2$ be two chain graphs. Given a subset $A_1 \subseteq V(G_1)$ and $A_2 \subseteq V(G_2)$, we say $(G_1, A_1)$ and $(G_2, A_2)$ are factorization-equivalent\(^2\) if they become the same chain graph after removing from $G_i$ all vertices in $A_i$ together with the edges incident to vertices in $A_i$ for $i \in \{1, 2\}$. Typically, $A_i$ is a set of constant variables in $V(G_i)$ created by intervention. Moreover, we say $(G_1, A_1), (G_2, A_2)$ admit the same factorization decomposition if every probability density $f$ that factorizes according to $G_1$ with $A_1$ intervened also factorizes according to $G_2$ with $A_2$ intervened, and vice versa.

Theorem 5.13. Let $G_1$ and $G_2$ be two chain graphs defined on the same set of variables $V$. Moreover a common set of variables $A$ in $V$ are set by intervention such that for every $a \in A$, $x_a = a_0$. If $(G_1, A)$ and $(G_2, A)$ are factorization-equivalent, then $G_1$ and $G_2$ admit the same factorization decomposition.

Proof. Let $G_0$ be the chain graph obtained from $G_1$ by removing all vertices in $A$ and the edges incident to $A$. It suffices to show that $G_1$ and $G_2$ both admit the same factorization decomposition as $G_0$. Let $D_1, D_0$ be the set of chain components of $G_1$ and $G_0$ respectively. Let $\tau \in D_1$ be an arbitrary chain component of $G_1$. By the factorization formula in (5.8), it follows that

$$f(x_{\tau \setminus A} \mid x_{pa(\tau)}, x_{\tau \cap A}) = Z^{-1}(x_{pa(\tau)}, x_{\tau \cap A}) \prod_{h \in C} \psi_h(x)$$

where $C$ is the set of maximal cliques in $(G_{\tau \cup pa(\tau)})^m$ and $Z^{-1}(x_{pa(\tau)}, x_{\tau \cap A}) = \int_{X_{\tau \setminus A}} \prod_{h \in C} \psi_h(x) \mu_{\tau \setminus A}(dx_{\tau \setminus A})$. Notice that for any maximal clique $h_1 \in C$ such that $h_1 \cap A = \emptyset$, $h_1$ is also a clique in $(G_0[\tau \setminus A])^m$. For $h_1 \in C$ with $h_1 \cap A \neq \emptyset$, there are two cases:

Case 1: $(h_1 \cap \tau) \setminus A \neq \emptyset$. In this case, observe that $h_1 \setminus A$ is also a clique in $(G_0[\tau \setminus A])^m$, thus is contained in some maximal clique $h'$ in $(G_0[\tau \setminus A])^m$. Since all variables in $A$ are pre-set as constants, it follows that $\psi_{h_1}(x)$ also appears in a factor in the factorization of $f$ according to $G_0$.

\(^2\)This term was defined for a different purpose in Studený, Roverato, and Š. Štěpánová (2009).
Case 2: $h_1 \cap \tau \subseteq A$. In this case, note that $h_1 \cap \tau$ is disjoint with $\tau \setminus A$. Hence $\psi_{h_1}(x)$ appears as a factor independently of $x_{\tau \setminus A}$ in both $Z^{-1}(x_{pa(\tau)}, x_{\tau \setminus A})$ and $\prod_{h \in C} \psi_{h}(x)$, which cancels out with itself.

Thus it follows that every probability density $f$ that factorizes according to $G_1$ also factorizes according to $G_0$. On the other hand, it is easy to see that for every $\tau' \in D_0$ and every maximal clique $h'$ in $(G_0[\tau'])^m$, $h'$ is contained in some maximal clique $h$ in $(G_1[\tau])^m$ for some $\tau \in D_1$. Hence we can conclude that $G_1$ and $G_0$ admit the same factorization decomposition. The above argument also works for $G_2$ and $G_0$. Thus, $G_1$ and $G_2$ admit the same factorization decomposition. 

We now define a graphical procedure (call it redirection procedure) that is consistent with the intervention formula in Equation (5.7) and (5.8). Let $G$ be a chain graph. Given an intervened set of variables $A \subseteq V(G)$, let $\hat{G}$ be the chain graph obtained from $G$ by performing the following operation: for every $u \in A$ and every undirected edge $e = \{u, w\}$ containing $u$, replace $e$ by a directed edge from $u$ to $w$; finally remove all the directed edges that point to some vertex in $A$. By replacing the undirected edge with a directed edge, we replace any feedback mechanisms that include a variable in $A$ with a causal mechanism.

The intuition behind the procedure is the following. Since a variable that is set by intervention cannot be modified, the symmetric feedback relation is turned into an asymmetric causal one. Similarly, we can justify this graphical procedure as equivalent to removing the variables in $A$ from some equations in the Gibbs process on top of p. 338 of (Lauritzen and Richardson, 2002), as Lauritzen and Richardson (Richardson, 2018) did for Equation (18) in (Lauritzen and Richardson, 2002).

**Theorem 5.14.** Let $G$ be a chain graph with a subset of variables $A \subseteq V(G)$ set by intervention such that for every $a \in A$. $x_a = a_0$. Let $\hat{G}$ be obtained from $G$ by the redirection procedure. Then $G$ and $\hat{G}$ admit the same factorization decomposition.
Proof. It is not hard to see that removing from \( \hat{G} \) and \( G \) all vertices in \( A \) and all edges incident to \( A \) results in the same chain graph. Hence by Theorem (5.13), \( G \) and \( \hat{G} \) admit the same factorization decomposition. \( \square \)

![Figure 5.12](image_url)

Figure 5.12: (a) A chain graph \( G \); (b) The graph \( \hat{G} \) obtained from \( G \) through the redirection procedure; (c) The graph \( G_0 \) obtained from \( G \) by deleting variables in \( A \).

**Example 22.** Consider the chain graph \( G \) shown in Figure 5.12. Let \( \hat{G} \) be the graph obtained from \( G \) through the redirection procedure described in this subsection. Let \( G_0 \) be the chain graph obtained from \( G \) by deleting the vertex \( c_0 \) and the edges incident to \( c_0 \). We will compare the factorization decomposition according to the formula (5.7),(5.8) as well as the graph structure \( \hat{G} \) and \( G_0 \).

By the formula (5.7) and (5.8) proposed in (Lauritzen and Richardson, 2002), when \( x_c \) is set as \( c_0 \) by intervention,

\[
f(x||x_c) = f(x_a)f(x_b)f(x_{de}|x_{abc_0})
\]

\[
= f(x_a)f(x_b) \frac{\psi_{ac0d}(x)\psi_{abde}(x)}{\sum_{d,e} \psi_{ac0d}(x)\psi_{abde}(x)}.
\]

Now consider the factorization according to \( \hat{G} \). The chain components of \( \hat{G} \) are \{\{a\}, \{b\}, \{c\}, \{d, e\}\} with \( x_c \) set to be \( c_0 \). The factorization according to \( \hat{G} \) is as follows:

\[
f_{\hat{G}}(x||x_c) = f_{\hat{G}}(x_a)f_{\hat{G}}(x_b)f_{\hat{G}}(x_c)f_{\hat{G}}(x_{de}|x_{abc_0})
\]

\[
= f_{\hat{G}}(x_a)f_{\hat{G}}(x_b)f_{\hat{G}}(x_c) \frac{\psi_{ac0d}(x)\psi_{abde}(x)}{\sum_{d,e} \psi_{ac0d}(x)\psi_{abde}(x)},
\]

where \( f(x_c) = 1 \) when \( x_c = c_0 \) and otherwise 0. Hence \( G \) and \( \hat{G} \) admit the same factorization.
Now consider the factorization according to $G_0$. The chain components of $G_0$ are $\{\{a\}, \{b\}, \{d, e\}\}$. The factorization according to $G_0$ is as follows:

$$f_0(x) = f_0(x_a)f_0(x_b)f_0(x_{de})|x_{ab} = \psi_{ad}(x)\psi_{abde}(x)\sum_{d,e} \psi_{ad}(x)\psi_{abde}(x),$$

Observe that $f_0(x)$ has the same form of decomposition as $f(x|x_c)$ since $x_c$ is set to be $c_0$ in $\psi_{ac0d}(x)$ (with the understanding that the probability of any configuration of variables with $x_c \neq c_0$ is zero). Hence we can conclude that $G, \hat{G}$ (with $x_c$ intervened) and $G_0$ admit the same factorization decomposition.

### 5.4.2 Factorization equivalence and intervention in Bayesian hypergraphs

Intervention in Bayesian hypergraphs can be modeled analogously to the case of chain graphs. We use the same notation as before. Let $\mathcal{H}$ be a DAH and $\{\tau: \tau \in \mathcal{D}\}$ be its chain components. Moreover, assume further that a subset $A$ of variables in $V(\mathcal{H})$ are set such that for every $a \in A$, $x_a = a_0$. Then a probability density $f$ factorizes according to $\mathcal{H}$ (with $A$ intervened) as follows: (where it is understood that the probability of any configuration of variables inconsistent with the intervention is zero):

$$f(x||x_A) = \prod_{\tau \in \mathcal{D}} f(x_{\tau\setminus A}|x_{pa(\tau)}, x_{\tau\cap A}).$$  \hspace{1cm} (5.9)

For each $\tau \in \mathcal{D}$, define $\mathcal{H}_{\tau}^*$ to be the subhypergraph of $\mathcal{H}_{\tau \cup pa(\tau)}$ containing all edges $h$ in $\mathcal{H}_{\tau \cup pa(\tau)}$ such that $H(h) \subseteq \tau$, then

$$f(x_{\tau\setminus A}|x_{pa(\tau)}, x_{\tau\cap A}) = Z^{-1}(x_{pa(\tau)}, x_{\tau\cap A}) \prod_{h \in M(\mathcal{H}_{\tau}^*)} \psi_h(x).$$  \hspace{1cm} (5.10)

where $Z^{-1}(x_{pa(\tau)}, x_{\tau\cap A}) = \int_{X_{\tau\setminus A}} \prod_{h \in M(\mathcal{H}_{\tau}^*)} \psi_h(x)\mu_{\tau\setminus A}(dx_{\tau\setminus A})$ and $\psi_h$ are non-negative functions that depend only on $x_h$.

**Definition 5.15.** Let $\mathcal{H}_1$ and $\mathcal{H}_2$ be two Bayesian hypergraphs. Given a subset of variables $A_1 \subseteq V(\mathcal{H}_1)$ and $A_2 \subseteq V(\mathcal{H}_2)$, we say $(\mathcal{H}_1, A_1)$ and $(\mathcal{H}_2, A_2)$ are factorization-equivalent
if performing the following operations to $\mathcal{H}_1$ and $\mathcal{H}_2$ results in the same directed acyclic hypergraph:

(i) Deleting all hyperedges with empty head, i.e., hyperedges of the form $(S, \emptyset)$.

(ii) Deleting every hyperedge that is contained in some other hyperedge, i.e., delete $h$ if there is another $h'$ such that $T(h) \subseteq T(h')$ and $H(h) \subseteq H(h')$.

(iii) Shrinking all hyperedges of $\mathcal{H}_i$ containing vertices in $A_i$, i.e. replace every hyperedge $h$ of $\mathcal{H}_i$ by $h' = (T(h) \setminus A_i, H(h) \setminus A_i)$ for $i \in \{1, 2\}$.

Typically, $A$ is a set of constant variables in $V$ created by intervention.

**Theorem 5.16.** Let $\mathcal{H}_1$ and $\mathcal{H}_2$ be two DAHs defined on the same set of variables $V$. Moreover, a common set of variables $A$ in $V$ are set by intervention such that for every $a \in A$, $X_a = a_0$. If $(\mathcal{H}_1, A)$ and $(\mathcal{H}_2, A)$ are factorization-equivalent, then $\mathcal{H}_1$ and $\mathcal{H}_2$ admit the same factorization decomposition.

**Proof.** In a way similar to what is done in the proof of Theorem 5.13, let $\mathcal{H}_0$ be the DAH obtained from $\mathcal{H}_1$ (or $\mathcal{H}_2$) by performing the operations above repeatedly. Let $\mathcal{D}_1$ and $\mathcal{D}_0$ be the set of chain components of $\mathcal{H}_1$ and $\mathcal{H}_0$ respectively. First, note that performing the operation $(i)$ does not affect the factorization since hyperedges of the form $h = (S, \emptyset)$ never appear in the factorization decomposition due to the fact that $H(h) \cap \tau = \emptyset$ for every $\tau \in \mathcal{D}_1$.

Secondly, $(ii)$ does not change the factorization decomposition too since if one hyperedge $h$ is contained in another hyperedge $h'$ as defined, then $\psi_h(x)$ can be simply absorbed into $\psi_{h'}(x)$ by replacing $\psi_{h'}(x)$ with $\psi_{h'}(x) \cdot \psi_h(x)$.

Now let $\tau \in \mathcal{D}_1$ be an arbitrary chain component of $\mathcal{H}_1$ and $h_1 \in \mathcal{H}_1[\tau]^+$, i.e., the set of hyperedges in $\mathcal{H}_1$ whose head intersects $\tau$. Suppose that $\tau$ is separated into several chain components $\tau'_1, \tau'_2, \cdots, \tau'_t$ in $\mathcal{H}_0$ because of the shrinking operation. If $h_1 \cap A = \emptyset$, then $h_1$ is also a hyperedge in $\mathcal{H}_0[\tau \setminus A]^+$. If $h_1 \cap A \neq \emptyset$, there are two cases:
Case 1: $H(h_1) \subseteq A$. Then since variables in $A$ are constants, it follows that in Equation (5.10), $\psi_{h_1}(x)$ does not depend on variables in $\tau \backslash A$. Hence $\psi_{h_1}(x)$ appears as factors independent of variables in $\tau \backslash A$ in both $Z^{-1}(x_{pa(\tau)}, x_{\tau \cap A})$ and $\prod_{h \in M(\mathcal{H}_0^\tau)} \psi_h(x)$, thus cancels out with itself. Note that, $h_1$ does not exist in $\mathcal{H}_0$ too since $h_1$ becomes a hyperedge with empty head after being shrinked and thus is deleted in Operation (i).

Case 2: $H(h_1) \backslash A \neq 0$. In this case, $H(h_1) \backslash A$ must be entirely contained in one of $\{\tau'_1, \cdots, \tau'_t\}$. Without loss of generality, say $H(h_1) \backslash A \subseteq \tau'_1$ in $\mathcal{H}_0$. Then note that $h_1 \backslash A$ must be contained in some maximal hyperedge $h'$ in $E(\mathcal{H}_0)$ such that $H(h') \cap \tau'_1 \neq \emptyset$. Moreover, recall that variables in $A$ are constants. Hence $\psi_{h_1}$ must appear in some factor in the factorization of $f$ according to $\mathcal{H}_0$.

Thus it follows that every probability density $f$ that factorizes according to $\mathcal{H}_1$ also factorizes according to $\mathcal{H}_0$. On the other hand, it is not hard to see that for every $\tau' \in \mathcal{D}_0$ and every hyperedge $h'$ in $(\mathcal{H}_0[\tau'])^*$, $h'$ is contained in some maximal hyperedge $h$ in $(\mathcal{H}_1[\tau])^*$ for some $\tau \in \mathcal{D}_1$. Hence we can conclude that $\mathcal{H}_1$ and $\mathcal{H}_0$ admit the same factorization decomposition. The above argument also works for $\mathcal{H}_2$ and $\mathcal{H}_0$. Thus, $\mathcal{H}_1$ and $\mathcal{H}_2$ admit the same factorization decomposition. \hfill \Box

We now present a graphical procedure (call it redirection procedure) for modeling intervention in Bayesian hypergraph. Let $\mathcal{H}$ be a DAH and $\{\tau : \tau \in \mathcal{D}\}$ be its chain components. Suppose a set of variables $x_A$ is set by intervention. We then modify $\mathcal{H}$ as follows: for each hyperedge $h \in E(\mathcal{H})$ such as $S = H(h) \cap A \neq \emptyset$, replace the hyperedge $h$ by $h' = (T(h) \cup S, H(h) \backslash S)$. If a hyperedge has empty set as its head, delete that hyperedge. Call the resulting hypergraph $\mathcal{H}_A$. We will show that the factorization according to $\mathcal{H}_A$ is consistent with Equation (5.10).

**Theorem 5.17.** Let $\mathcal{H}$ be a Bayesian hypergraph and $\{\tau : \tau \in \mathcal{D}\}$ be its chain components. Given an intervened set of variables $x_A$, let $\mathcal{H}_A$ be the DAH obtained from $\mathcal{H}$ by replacing each hyperedge $h \in E(\mathcal{H})$ satisfying $S = H(h) \cap A \neq \emptyset$ by the hyperedge $h' = (T(h) \cup \emptyset$.
S, H(h)\S) and removing hyperedges with empty head. Then \( \mathcal{H} \) and \( \hat{\mathcal{H}} \) admit the same factorization decomposition.

**Proof.** This is a corollary of Theorem (5.16) since performing the operations (i)(ii)(iii) in the definition of factorization-equivalence of DAH to \( \mathcal{H} \) and \( \hat{\mathcal{H}} \) results in the same DAH. \( \square \)

![Diagram](image)

**Example 23.** Let \( G \) be a chain graph as shown in Figure 5.13(a) and \( \mathcal{H} \) be the canonical LWF Bayesian hypergraph of \( G \) as shown in Figure 5.13(b), constructed based on the procedure in Section 5.1.4. \( \mathcal{H} \) has two directed hyperedges (\( \{a\}, \{c, d\} \)) and (\( \{a, b\}, \{d, e\} \)). Applying the redirection procedure for intervention in Bayesian hypergraphs leads to the Bayesian hypergraph \( \hat{\mathcal{H}} \) in Figure 5.13(c). We show that using equations (5.7) and (5.8) for Figure 5.13(a) leads to the same result as if one uses the factorization formula for the Bayesian hypergraph in Figure 5.13(c).

First, we compute \( f(x|X_c) \) for chain graph in Figure 5.13(a). Based on equation (5.7) we have:

\[
f(x|X_c) = f(x_a)f(x_b)f(x_{de}|x_{abc}),
\]

as the effect of the atomic intervention \( do(X_c = c_0) \). Then, using equation (5.8) gives:

\[
f(x|X_c) = f(x_a)f(x_b)\frac{\psi_{ac_0d}(x)\psi_{abde}(x)}{\sum_{d,e} \psi_{ac_0d}(x)\psi_{abde}(x)}. \tag{5.11}
\]
Now, we compute \( f(x) \) for Bayesian hypergraph in Figure 5.13(c). Using equation (5.4) gives:

\[
f(x||x_c) = f(x_a)f(x_b)f(x_{de} | x_{abc_0}).
\]

Applying formula (5.5) gives:

\[
f(x||x_c) = f(x_a)f(x_b)f(x_c)\frac{\psi_{acod}(x)\psi_{abde}(x)}{\sum_{d,e} \psi_{acod}(x)\psi_{abde}(x)} \tag{5.12}
\]

Note that \( f(x) = 1 \), when \( x_c = c_0 \), otherwise \( f(x_c) = 0 \). As a result, the right side of equations (5.11) and (5.12) are the same.

![Figure 5.14: Commutative diagram of factorization equivalence](image)

**Remark 14.** Figure 5.14 summarizes all the results in Section 5.4. Given a chain graph \( G \) and its canonical LWF DAH \( \mathcal{H} \), Theorem 5.11 shows that \( G \) and \( \mathcal{H} \) admit the same factorization decomposition. Suppose a set of variables \( A \) is set by intervention. Theorem 5.13 and 5.14 show that the the DAH obtained from \( G \) by the redirection procedure or deleting the variables in \( A \) admit the same factorization decomposition, which is also consistent with the intervention formula introduced in (Lauritzen and Richardson, 2002). Similarly,
Theorem 5.16 and 5.17 show that the DAH obtained from $H$ by the redirection procedure or shrinking the variables in $A$ admit the same factorization decomposition, which is consistent with a hypergraph analogue of the formula in (Lauritzen and Richardson, 2002).

CONCLUSION AND FUTURE WORK

This chapter presents Bayesian hypergraph, a new probabilistic graphical model. We showed that the model generalizes Bayesian networks, Markov networks, and LWF chain graphs, in the following sense: when the shadow of a Bayesian hypergraph is a chain graph, its Markov properties are the same as that of its shadow. We extended the causal interpretation of LWF chain graphs to Bayesian hypergraphs and provided corresponding formulas and two graphical procedures for intervention (as defined in (Lauritzen and Richardson, 2002)).

Directed acyclic hypergraphs can admit much finer factorizations than chain graphs, thus are more computationally efficient. The Bayesian hypergraph model also allows simpler and more general procedures for factorization as well as intervention. Furthermore, it allows a modeler to express independence of causal influence and other useful patterns, such as Noisy-OR, directly (i.e., graphically), rather than through annotations or the structure of a conditional probability table or function. We conjecture that the greater expressive power of Bayesian hypergraphs can be used to represent other PGMs and plan to explore the conjecture in future work.

Although the Bayesian hypergraph model admits much finer factorizations than the chain graph model, we note that our model does not yet expand the set of independence lists that can be represented by chain graphs. This is due to the fact that the Markov properties of the Bayesian hypergraphs are defined based on the shadow of the Bayesian hypergraph (which is a chain graph). One direction for future work is to design a hypergraph graphical model that admits a much larger set of independence lists than those allowed by chain graphs. Learning the structure and the parameters of Bayesian hypergraphs is
another direction for future work. For this purpose, we will need to provide a criterion for Markov equivalence of Bayesian hypergraphs. The success of constraint-based structure learning algorithms for chain graphs leads us to hope that similar techniques would work for learning Bayesian hypergraphs. Of course, one should also explore whether a closed-form decomposable likelihood function can be derived in the discrete finite case.
Modern systems (e.g., deep neural networks, big data analytics, and compilers) are highly configurable, which means they expose different performance behavior under different configurations. The fundamental challenge is that one cannot simply measure all configurations due to the sheer size of the configuration space. Transfer learning has been used to reduce the measurement efforts by transferring knowledge about performance behavior of systems across environments. Previously, research has shown that statistical models are indeed transferable across environments. In this chapter, we investigate identifiability and transportability of causal effects and statistical relations in highly-configurable systems. Our causal analysis agrees with previous exploratory analysis (Jamshidi et al., 2017) and confirms that the causal effects of configuration options can be carried over across environments with high confidence. We expect that the ability to carry over causal relations will enable effective performance analysis of highly-configurable systems.

6.1 Introduction

To understand and predict the effect of configuration options in configurable systems, different sampling and learning strategies have been proposed (Siegmund et al., 2015; Valov et al., 2017; Sarkar et al., 2015), albeit often with significant cost to cover the highly dimensional configuration space. Recently, we performed an exploratory analysis to understand why and when transfer learning works for configurable systems (Jamshidi et al., 2017). In this paper, instead of statistical analysis, we employ causal analysis to address the possibility of identifying influential configuration options that have a causal relation with the
performance metrics of configurable systems (identifiability) and whether such causal relations are transferable across environments (transportability).

![Diagram of causal inference for performance analysis](image)

Figure 6.1: Exploiting causal inference for performance analysis.

Recently, transfer learning has been used to decrease the cost of learning by transferring knowledge about performance behavior across environments (Jamshidi et al., 2018; Valov et al., 2017). Fortunately, performance models typically exhibit similarities across environments, even environments that differ substantially in terms of hardware, workload, or version (Jamshidi et al., 2017). The challenge is to (i) identify similarities and (ii) make use of them to ease learning of performance models.

To estimate causal effects, scientists normally perform randomized experiments where a sample of units drawn from the population of interest is subjected to the specified manipulation directly. In many cases, however, such a direct approach is not possible due to expense or ethical considerations. Instead, investigators have to rely on observational studies to infer effects. One of the fundamental questions in causal analysis is to determine when effects can be inferred from statistical information, encoded as a joint probability dis-
tribution, obtained under normal, intervention-free measurement. Pearl and his colleagues have made major contributions in solving the problem of identifiability. Pearl (Pearl, 1995) established a calculus of interventions known as *do-calculus*, consisting of three inference rules by which probabilistic equations involving interventions and observations can be transformed into other such equations, thus providing a syntactic method of deriving claims about interventions. Later, do-calculus was shown to be complete for identifying causal effects, that is, every causal effect that can be identified can be derived using the three do-calculus rules (Huang and Valtorta, 2006a,b, 2008; Shpitser and Pearl, 2006a,b).

Pearl and Bareinboim (Pearl and Bareinboim, 2011; Bareinboim and Pearl, 2012; Pearl and Bareinboim, 2014; Bareinboim and Pearl, 2016) provided strategies for inferring information about new populations from trial results that are more general than re-weighting. They supposed that we have available both causal information and probabilistic information for population $A$ (i.e., the source), while for population $B$ (i.e., the target) we have only (some) probabilistic information, and also that we know that certain probabilistic and causal facts are shared between the two and certain ones are not. They offered theorems describing what causal conclusions about population $B$ are thereby fixed. Conclusions about one population can be supported by information about another depends on exactly what causal and probabilistic facts they have in common.

In this chapter, we conduct a *causal analysis*, comparing performance behavior of highly-configurable systems across environmental conditions (changing workload, hardware, and software versions), to explore when and how causal knowledge can be commonly exploited for performance analysis. In this paper, we use the proposed formal language of causal graphs for identifiability and transportability in the literature, to answer:

Is it possible to identify causal relations from observational data and how generalizable are they in highly-configurable systems?

Our results indicate the possibility of identifiability of causal effects in general. Also, our results show that many of causal/statistical relations about performance behavior can...
be transferred across environments even in the most severe changes we explored, and that transportability is actually trivial for many environmental changes. Our empirical results also indicate the recoverability of conditional probabilities from selection-biased data in many cases. The results indicate that causal information can be used as a guideline for cost-efficient sampling for performance prediction of configurable systems. The supplementary materials including data and empirical results are available at: https://github.com/majavid/AAAI-WHY-2019.

6.2 Causal Graphs

A causal graphical model is a special type of Bayesian network in which edges are interpreted as direct causal effects. This interpretation facilitates predictions under arbitrary (unseen) interventions, and hence the estimation of causal effects (Pearl, 2009). In this section, we consider two constraint-based methods to estimate the causal structure from observational data. For this purpose, we discuss the PC algorithm and the fast causal inference (FCI) algorithm (Spirtes, Glymour, and Scheines, 2000).

6.2.1 Estimating causal structures

A causal structure without feedback loops and without hidden or selection variable can be visualized using a directed acyclic graph (DAG) where the edges indicate direct cause-effect relationships. Under some assumptions, Pearl (Pearl, 2009) showed that there is a link between causal structures and graphical models. Roughly speaking, if the underlying causal structure is a DAG, we observe data generated from this DAG and then estimate a DAG model (i.e., a graphical model) on this data, the estimated complete partially directed acyclic graph (CPDAG) represents the equivalence class of the DAG model describing the causal structure. This holds if we have enough samples and assuming that the true underlying causal structure is indeed a DAG without unobserved common causes (confounders) or selection variables. Note that even given an infinite amount of data, we usually cannot
identify the true DAG itself, but only its equivalence class. Every DAG in this equivalence class can be the true causal structure (Kalisch et al., 2012).

In the case of unobserved variables, one could still visualize the underlying causal structure with a DAG that includes all observed, unobserved cause, and unobserved selection variables. However, when inferring the DAG from observational data, we do not know all unobserved variables. We, therefore, seek to find a structure that represents all conditional independence relationships among the observed variables given the selection variables of the underlying causal structure. It turns out that this is possible. However, the resulting object is in general not a DAG for the following reason. Suppose, we have a DAG including observed and unobserved variables, and we would like to visualize the conditional independencies among the observed variables only. We could marginalize out all unobserved cause variables and condition on all unobserved selection variables. It turns out that the resulting list of conditional independencies can in general not be represented by a DAG, since DAGs are not closed under marginalization or conditioning (Richardson and Spirtes, 2002). A class of graphical independence models that is closed under marginalization and conditioning and that contains all DAG models is the class of ancestral graphs (Richardson and Spirtes, 2002). A mixed graph is a graph containing three types of edges, undirected (−), directed (→) and bidirected (↔). An ancestral graph $G$ is a mixed graph in which the following conditions hold for all vertices in $G$:

(i) if $\alpha$ and $\beta$ are joined by an edge with an arrowhead at $\alpha$, then $\alpha$ is not anterior to $\beta$.

(ii) there are no arrowheads present at a vertex which is an endpoint of an undirected edge.

**Maximal ancestral graphs** (MAGs), which we will use from now on, also obey a third rule:

(iii) every missing edge corresponds to a conditional independence.

An equivalence class of a MAG can be uniquely represented by a *partial ancestral graph* (PAG) (Zhang, 2008). Edge directions are marked with “−” and “>” if the direction is the
same for all graphs belonging to the PAG and with “o” otherwise. The bidirected edges come from hidden variables, and the undirected edges come from selection variables.

We use the Hugin PC algorithm and the FCI algorithm in the R package pcalg to recover the causal graph of each environment for our subject systems. Since all possible configurations of options are present in the first and last subject systems in Table 6.1 and all data sets have been sampled on the basis of configuration settings alone, we can assume that there are no unobserved common causes and selection variables, i.e., the causal sufficiency assumption (Spirtes, Glymour, and Scheines, 2000) holds. In other cases, due to the sparsity of data, we cannot exclude the presence of hidden variables, therefore, we use the FCI algorithm to recover the causal graphs.

6.3 Research Questions and Methodology

The overall question that we explore in this paper is “why and when identifiability and transportability of causal effects can be exploited in configurable systems?” We hypothesize that estimating causal effects from observational studies alone, without performing randomized experiments or manipulations of any kind (causal inference of this sort is called identification (Pearl, 2009)) is possible for configurable software systems. Also, we speculate that causal relations in the source and the target are somehow related. To understand the notion of identification and relatedness that we find for environmental changes, we explore three questions.

**RQ1.** Is it possible to estimate causal effects of configuration options on performance from observational studies alone?

If we can establish with RQ1 that causal effects of configuration options on the performance are estimable, this would be promising for performance modeling in configurable systems because it helps us to estimate an accurate, reliable, and less costly causal effect in an environment. Even if not all causal effects may be estimable, we explore which configuration options are influential on performance.
RQ2. Is the causal effect of configuration options on performance transportable across environments?

RQ2 concerns transferable knowledge from the source that can be exploited to learn an accurate and less costly performance model for the target environment. Specifically, we explore how the causal effects of influential options are transportable across environments and how they can be estimated.

RQ3. Is it possible to recover conditional probabilities from selection-biased data to the entire population?

RQ3 concerns transferable knowledge that can be exploited for recovering conditional probabilities from selection-biased data to the population. Specifically, we explore whether causal/statistical relations between configuration options and performance measures are recoverable from a biased sample without resorting to external information.

6.3.1 Methodology

Design: We investigate the causal effects of configuration options on performance measures across environments. So, we need to establish the performance of a system and how it is affected by configuration options in multiple environments. As in (Jamshidi et al., 2017), we measure the performance of each system using standard benchmarks and repeat the measurements across a large number of configurations. We then repeat this process for several changes to the environment: using different hardware, workloads, and versions of the system. Finally, we perform the analysis of relatedness by comparing the performance and how it is affected by options across environments. We perform comparison of a total of 65 environment changes.

Analysis: For answering the research questions, we formulate three hypotheses about:

- **Identifiability**: The causal effect of $X$ on $Y$ is identifiable from a causal graph $G$ if the quantity $P(y|do(x))$ can be computed uniquely from any positive probability of the
observed variables (Pearl, 2009).

- **Transportability**: Given two environments, denoted Π and Π∗, characterized by probability distributions $P$ and $P^*$, and causal diagrams $G$ and $G^*$, respectively, a causal relation $R$ is said to be transportable from Π to Π∗ if $R(\Pi)$ is estimable from the set $I$ of interventions on Π, and $R(\Pi^*)$ is identified from $P, P^*, I, G,$ and $G^*$ (Pearl and Bareinboim, 2011).

- **Recovering conditional probabilities**: Given a causal graph $G_s$ augmented with a node $S$ encoding the selection mechanism, the distribution $Q = P(y|x)$ is said to be $s$-recoverable from selection-biased data in $G_s$ if the assumptions embedded in the causal model renders $Q$ expressible in terms of the distribution under selection bias $P(v|S = 1)$ (Bareinboim, Tian, and Pearl, 2014).

For each hypothesis, we recover the corresponding causal graph and analyze 65 environment changes in four subject systems mentioned below. For each hypothesis, we discuss how commonly we identify this kind of estimation and whether we can identify classes of changes for which this estimation is characteristic. If we find out that for an environmental change a hypothesis holds, it means that enough knowledge is available to estimate causal effects/conditional probabilities across environments.

### 6.3.2 Subject systems

In this study, we selected four configurable software systems from different domains, with different functionalities, and written in different programming languages (Table 6.1). Further details can be found in (Jamshidi et al., 2017).

### 6.4 Identification of Causal Effects (RQ1)

We can derive a complete solution to the problem of identification whenever assumptions are expressible in a DAG form. This entails (i) graphical and algorithmic criteria for decid-
ing identifiability of causal effects, (ii) automated procedures for extracting all identifiable estimand (Pearl, 1995; Huang and Valtorta, 2006a; Shpitser and Pearl, 2006a).

Table 6.1: Overview of the real-world subject systems

| System    | Domain          | $d$  | $|C|$ | $|H|$ | $|W|$ | $|V|$ |
|-----------|-----------------|------|------|------|------|------|
| SPEAR     | SAT solver      | 14   | 16384| 3    | 4    | 2    |
| SQLite    | Database        | 14   | 1000 | 2    | 14   | 2    |
| x264      | Video encoder   | 16   | 4000 | 2    | 3    | 3    |
| XGBoost   | Machine learning| 12   | 4096 | 3    | 3    | 1    |

Here, we investigate the possibility of estimating causal effects of configuration options on performance from observational studies alone. For this purpose, we consider a hypothesis about the possibility of identifiability in experiments with a single performance metric (e.g., response time) and multiple performance metrics (e.g., response time and throughput). We expect that this hypothesis hold for (almost) all cases, which would enable an easy estimation of causal effects from the available data.

**H1**: The causal effect of options $O_i$ on performance $perf$ from observed data is identifiable.

**Importance**: If the causal effect of configuration options on performance is identifiable from available data, we can predict the performance behavior of a system in the presence/absence of a configuration option just by available observational data. Also, we may get rid of the curse of dimensionality in highly configurable systems to run and test new experiments. Because the recovered causal structure from the observed data indicates whether a given configuration option is influential on performance.

**Methodology**: We evaluate whether $P(perf|do(O_i = o'))$ is identifiable. We used PC or FCI algorithms (with two commonly used p-values .01 and 0.05) along with a set of background knowledge (came from experts’ opinions) that explains the observed independence facts in a sample, to learn the corresponding causal graph. For example, Figure 6.2 shows the obtained causal graph for x264 in the corresponding environment. We use this causal
Figure 6.2: Causal graph for x264 deployed on internal server Feature1 and used version 2.76.2 of x264 and used a small video for encoding. For all figures we do not show options that do not affect on performance.

Figure 6.3: Causal graph for XGBoost12 with CNAE-9 data set, deployed on Feature 4. Performance nodes are: train-time, test-time, and accuracy.

graph to estimate the causal effect of the configuration option visualize on the encoding time of the system i.e., \( P(\text{encoding} - \text{time} - \text{feature}1 - 2762 - 8|\text{do}(\text{visualize})) \). Also, Figure 6.3 shows the obtained causal graph for XGBoost12 in the corresponding environment.

We use this causal graph to estimate \( P(\text{test} - \text{time}|\text{do}(\text{max} - \text{depth})) \).

**Results:** First, the obtained causal graph in each case indicates which configuration options are influential on performance for the corresponding environment. In all instances (see supplementary material), the number of configuration options that affect the corresponding performance metric is remarkably small (usually less than 6), indicating that the dimensionality of the configuration space for sampling and running new experiments can be reduced drastically. This observation confirms the exploratory analysis in (Jamshidi et al., 2017), showing that only a small proportion of possible interactions have an effect on performance and so are relevant. For example, Figure 6.2 shows that only four (out of 16) configuration options effect the encoding time in the corresponding environment. Second, \( P(\text{perf}|\text{do}(O_i = o')) \) is estimable in all environments with a single mea-
surement, because in all cases, the pre-intervention and post-intervention (Pearl, 2009) causal graphs are the same, and so \( P(\text{perf}|\text{do}(O_i = o')) = P(\text{perf}|O_i = o') \), indicating that the hypothesis H1 holds in general. For example, for x264 deployed on internal server Feature1 and used version 2.76.2 of x264 and used a small video for encoding, using do-calculus and Hugin gives: \( P(\text{encoding} - \text{time} - \text{feature}1 - 2762 - 8|\text{do(visualize)} = 1) = P(\text{encoding} - \text{time} - \text{feature}1 - 2762 - 8|\text{visualize} = 1) \) with the mean of 0.37 and a variance of 0.14. Also, Figure 6.3 shows those configuration options that affect performance nodes in the corresponding environment. Similarly, we observed that \( P(\text{perf}|\text{do}(O_i = o')) \) is estimable in all environments with multiple measurements. For example, for XGBoost12, using Rule 2 of do-calculus gives: \( P(\text{test-time}|\text{do}(\text{max-depth})) = P(\text{test-time} | \text{max-depth}) \).

**Implications:** The results indicate that such information can be used to find (causal) influential options, leading to effective exploration strategies.

6.5 **Transportability of Causal and Statistical Relations Across Environments (RQ2)**

Here, we investigate the possibility of transportability of causal effects across environments. For this purpose, we consider a hypothesis about the possibility of transportability of causal/statistical relations across environments. We observed that this hypothesis holds for some cases with both small and even severe environmental changes, which would enable an easy generalization (trivial transportability\(^1\)) of causal and statistical relations from source to the target environment.

**H2:** The causal/statistical relation \( R \) is transportable across environments.

**Importance:** When experiments cannot be conducted in the target environment, and despite severe differences between the two environments, it might still be possible to compute causal relations by borrowing experimental knowledge from the source environment. Also,\(^1\)

\(^1\)This kind of transportability allows us to estimate causal/statistical relations directly from passive observations on the target environment, un-aided by causal/statistical information from the source environment Pearl and Bareinboim (2011).
if transportability is feasible, the investigator may select the essential measurements in both experimental and observational studies, and thus minimize measurement costs.

**Methodology:** We investigate whether $P(\text{perf} \mid \text{do}(O_i = o'))$ (or $P(\text{perf} \mid O_i = o')$) is transportable across environments. For this purpose, we first recover the corresponding causal graphs for source and target environments in a similar way to that described in H1. Since the S-variables in the selection diagram\(^2\) locate the mechanisms where structural discrepancies between the two environments are suspected to take place, we only add the selection node to the measurement metric node(s). For example, Figure 6.4 shows the selection diagram for SPEAR deployed on two different environments. We use this selection diagram to verify the transportability of $P(\text{perf} \mid \text{do}(\text{spset-hw-bmc}))$ and $P(\text{perf} \mid \text{spset-hw-bmc})$ across mentioned environments. Also, Figure 6.5 shows the obtained selection diagram for XGBoost\(^1\) in two environments. We use this selection diagram to verify the transportability of $P(\text{test-time} \mid \text{do}(\text{colsample-bylevel}))$.

**Results:** We observed that H2 holds for those environments (with single measurement metric) that share the same causal graph while the presence of a selection node pointing to the variable, say perf, in the selection diagram indicates that the local mechanism that assigns values to perf may not the same in both environments. In these cases, the corresponding selection diagram is $O_i \rightarrow \text{perf} \leftarrow S$, and so the causal/statistical relation is trivially transportable Pearl and Bareinboim (2011). This observation is consistent with the exploratory

\(^2\)A selection diagram is a causal diagrams augmented with a set, S, of "selection variables," where each member of S corresponds to a mechanism by which the two domains differ Pearl and Bareinboim (2011).
Figure 6.5: Selection diagram for XGBoost12 deployed on two environments: one deployed on a private server Feature 4, with covtype dataset, and another with the same characteristics but deployed on Azure Cloud. Performance nodes are: train-time, test-time, and accuracy.

analysis in Jamshidi et al. (2017), showing that for small environmental changes, the overall performance behavior is transportable across environments. However, our observations show that despite glaring differences between the two environments, it might still be possible to infer causal effects/statistical relations across environments. Also, we observed that transportability of causal/statistical relations across environments with multiple measurement metrics. In such cases, the complete algorithm in Bareinboim and Pearl (2012) can be used to derive the transport formula. Nevertheless, our observations indicate that transportable causal/statistical relations are trivial. For example, based on Figure 6.5, we have:

\[ P(\text{test-time}|\text{do(colsample-bylevel)}) = P(\text{test-time}|\text{colsample-bylevel}) \]

**Implications:** Transportability of causal relations can be exploited to avoid running new costly experiments in the target environment.

6.6 Generalizing Statistical Findings Across Sampling Conditions (RQ3)

Here, we examine the possibility of recovering conditional probabilities from selection-biased data. We consider a hypothesis about the possibility of recoverability without external data. We observed that this hypothesis holds for some cases, thus enabling the estimation of causal/statistical relations from selection-biased data to the entire population.

**H3:** The causal relations from selection-biased data are transportable to the population.
Importance: Since selection bias challenges the validity of inferences in statistical analysis, we may get rid of selection bias and estimate the causal/statistical relations of the entire population without resorting to external information.

Methodology: We use the causal graph $G_s$ augmented with a node $S$ that encodes the selection mechanism. According to Theorem 1 in Bareinboim, Tian, and Pearl (2014), the distribution $P(y|x)$ is $s$-recoverable from $G_s$ if and only if $S \perp\!\!\!\!\!\perp Y|X)$, which is a powerful test for $s$-recoverability.

Results: As we observed, in most cases, the recovered causal graph by FCI algorithm does not contain a non-chordal undirected component, indicating that FCI has not detected any selection bias from sampled data. In such cases, $s$-recoverability is the same as transportability. So, H3 holds for many cases in our study. For example, $P(fillseq|sqlite – omit – quickbalance)$ is not $s$-recoverable in Figure 6.6 (a) and (c), but it is $s$-recoverable in Figure 6.6 (b) and (d). In the data collected for the performance analysis of configurable systems, authors of Jamshidi et al. (2017, 2018) sampled on the basis of configuration settings alone; therefore the conditions of Figure 6.6 (b) and (d) hold, i.e., the selection bias is benign and the distribution of performance given configuration settings is recoverable. In these cases, knowledge from a sampled subpopulation can be generalized to the entire population. However, FCI recovered some structures of the type of Figure 6.6 (a), indicating that the sample size is small enough that some (implicit) selection bias connecting performance with one or more configuration settings.

Implications: Causal information can be used as a guideline for cost-efficient sampling for performance prediction of configurable systems and avoiding of biased estimates of causal/statistical effects in cases that recoverability was not possible.

6.7 Threats to Validity

1) External validity: We selected a diverse set of subject systems and a large number of purposefully selected environment changes, but, as usual, one has to be careful when gen-
eralizing to other subject systems and environmental changes.

2) Internal and construct validity: Due to the size of configuration spaces, we could only measure configurations exhaustively in two subject systems and had to rely on sampling (with substantial size) for the others, which may miss causal effects in parts of the configuration space that we did not sample.

CONCLUSION AND FUTURE WORK

To the best of our knowledge, this is the first paper that uses causal analysis to identify the key knowledge pieces that can be exploited for transfer learning in highly-configurable systems. Our empirical study demonstrates the existence of diverse forms of transferable causal effects across environments that can contribute to learning faster, better, reliable, and more importantly, less costly performance behavior analysis in configurable systems. For a future research direction, it would be interesting to explore how causal analysis can be employed for developing effective sampling methods and provide explainable performance analysis in configurable systems.
CHAPTER 7

CONCLUDING REMARKS

Graphical models provide a strong and clear formalism for studying conditional independence relations, probabilistic reasoning, and decision making that arise in different research areas. Originally, graphs with a single type of edge were used i.e., undirected graphs and DAGs. However, in the case of undirected graphs only symmetric relations i.e., correlation between variables can be represented and in the case of DAGs only asymmetric relations i.e., cause and effect relation between variables can be represented.

Chain graphs were introduced as a unification of directed and undirected graphs to model systems containing both symmetric and asymmetric relations. However, with the introduction of chain graphs (Lauritzen and Wermuth, 1989) as well as different interpretations of chain graphs (Andersson, Madigan, and Perlman, 1996; Cox and Wermuth, 1993), a very large number of Markov properties have emerged. This has led to confusion regarding Markov properties of chain graphs, especially in the case of MVR chain graphs.

In this thesis we therefore studied some of fundamental concepts regarding chain graphs (e.g., Markov properties and minimal separators) and in the case of MVR CGs we showed under which conditions proposed Markov properties for them are equivalent. In addition, we presented an order-independent PC-like algorithm for learning the structure of all three different interpretations of chain graphs and two decomposition-based algorithms, one for learning the structure of AMP CGs and another for learning the structure of MVR CGs from sampled data under the faithfulness assumption.

Although chain graphs are considerably more expressive than DAGs and undirected graphs i.e., the space of independence models representable by CGs is much larger than
DAGs and undirected graphs (Sonntag, Peña, and Gómez-Olmedo, 2015), there are many independence models that cannot be represented by CGs. For this purpose, we proposed a directed acyclic hypergraph framework for a novel probabilistic graphical model that we call Bayesian hypergraphs. The space of directed acyclic hypergraphs is much larger than the space of chain graphs. Hence Bayesian hypergraphs can model much finer factorizations than Bayesian networks or LWF chain graphs and provide simpler and more computationally efficient procedures for factorizations and interventions.

Using an expressive PGM class has both advantages and disadvantages. The main advantage is that a model of a more expressive class is more likely to capture the true relations between the variables in the system while less expressive classes make assumptions like for example that only causal relations exist between variables. The disadvantage of using an expressive class is that it can be harder to find the correct model since the number of possible models is much larger. This also makes it easier to overfit the learning data. Hence, to get an accurate model, more data is generally needed when learning expressive PGM classes compare to less expressive classes (Sonntag, 2014). Graphs with multiple types of edges can also be harder to interpret since the interpretation of what an edge represents is not always clear. For example, see (Lauritzen and Richardson, 2002) for the valid interpretation of LWF CGs. In addition, the more basic classes, such as Bayesian networks and Markov networks, have received more attention in research and hence more efficient learning and inference algorithms exist for these compared to the more general classes.

One important question when discussing different PGM classes is why CGs or Bayesian hypergraphs are interesting when there exist more general and expressive PGM classes such as marginal AMP graphs (Peña, 2014b) or ancestral graphs (Richardson and Spirtes, 2002)? The more expressive PGMs are, the more difficult the corresponding concepts and problems become. For example, though necessary and sufficient graphical conditions under which two ancestral graphs are Markov equivalent had been given previously in (Ali and Richardson, 2002) and (Zhao, Zheng, and Liu, 2005), the criterion which leads to an algo-
Algorithm that runs in polynomial time based on the size of the graph was provided later in (Ali, Richardson, and Spirtes, 2009). Furthermore, studying less expressive PGMs such as CGs can be used as a guideline to find out whether concepts and theoretical results regarding BNs can be extended to more general and expressive PGMs such as marginal AMP graphs and ancestral graphs. For example, (Peña, Sonntag, and Nielsen, 2014) developed an algorithm for learning LWF CGs under the composition property. However, (Peña, 2014a) proved that the same technique cannot be used for MVR chain graphs. Since, MVR CGs are a subclass of ancestral graphs (Theorem 3.4), we conclude that the same approach used in (Peña, Sonntag, and Nielsen, 2014) cannot be used for learning the structure of ancestral graphs from sampled data.

Table 7.1 summarizes some of the most important attributes of different types of common interpretations of chain graphs.

To address the applicability of graphical models, we conducted a causal analysis, comparing performance behavior of highly-configurable systems across environmental conditions (changing workload, hardware, and software versions), to explore when and how causal knowledge can be commonly exploited for performance analysis.

There are many tasks that were not investigated in this thesis and could benefit from the results obtained here. For example, the natural continuation of the work presented here would be to develop a learning algorithm via decomposition for ancestral graphs and marginal AMP graphs under the faithfulness assumption. Learning the structure and the parameters of Bayesian hypergraphs is another direction for future work. The success of constraint-based structure learning algorithms for chain graphs leads us to hope that similar techniques would work for learning Bayesian hypergraphs. Since all of proposed algorithms in the literature for learning CGs are constraint-based algorithms, another important area for future research is designing efficient scoring functions and score-based learning algorithms for CGs.
<table>
<thead>
<tr>
<th>Type of chain graph</th>
<th>Causal interpretation</th>
<th>Global Markov property</th>
<th>Factorization of $p(x)$</th>
<th>Model selection (structural learning) algorithm(s) [constraint based method]</th>
</tr>
</thead>
<tbody>
<tr>
<td>MVR CGs: Cox &amp; Wermuth (Cox and Wermuth, 1993, 1996; Wermuth and Cox, 2004), Peña &amp; Sonntag (Sonntag and Peña, 2015a; Sonntag and Peña, 2015b; Sonntag, 2014), Sadeghi &amp; Lauritzen (Sadeghi and Lauritzen, 2014), Drton (type IV) (Drton, 2009), Marchetti &amp; Lupparelli (Marchetti and Lupparelli, 2008, 2011)</td>
<td>Directed edges represent direct causal effects. Any bidirected edge $X \leftrightarrow Y$ represents one or more hidden common causes between $X$ and $Y$ (claimed in (Cox and Wermuth, 1996; Wermuth and Sadeghi, 2012; Sadeghi and Lauritzen, 2014; Sonntag, 2014), proved in Corollary 3.5)</td>
<td>(1) $X \perp ! ! \perp Y</td>
<td>Z$ if $X$ is separated from $Y$ by $Z$ in $(G_{\text{MVR}(X,Y,Z)})^<em>$ or $(G_{\text{MVR}(X,Y,Z)})^</em>$ (Richardson and Spirtes, 2002; Richardson, 2003). (2) $X \perp ! ! \perp Y</td>
<td>Z$ if $X$ is separated from $Y$ by $Z$ in $(G_{\text{MVR}(X,Y,Z)})^*$. (1) and (2) are equivalent (see Theorem 3.15).</td>
</tr>
<tr>
<td>LWF CGs (Freydenberg, 1990; Lauritzen and Wermuth, 1989), Drton (type I) (Drton, 2009)</td>
<td>Directed edges represent direct causal effects. Undirected edges represent causal effects due to interference (Shpitser, Tchetgen, and Andrews, 2017; Ogburn, Shpitser, and Lee, 2018; Richardson and Spirtes, 2002).</td>
<td>$X \perp ! ! ! \perp Y</td>
<td>Z$ if $X$ is separated from $Y$ by $Z$ in $(G_{\text{LWF}(X,Y,Z)})^*$ (Lauritzen, 1996).</td>
<td>(Cowell et al., 1999; Lauritzen and Richardson, 2002) $\prod_{T \in \mathcal{T}} p(x_T</td>
</tr>
<tr>
<td>AMP CGs (Andersson, Madigan, and Perlman, 1996), Drton (type II) (Drton, 2009)</td>
<td>Every AMP CG is Markov equivalent to some DAG with error and selection nodes under marginalization of the error (deterministic) nodes and conditioning of the selection nodes (Peña, 2014b).</td>
<td>$X \perp ! ! ! \perp Y</td>
<td>Z$ if $X$ is separated from $Y$ by $Z$ in the undirected graph $\text{Aug}(\text{CG}, X, Y, Z)$ (Andersson, Madigan, and Perlman, 1998).</td>
<td>$\prod_{T \in \mathcal{T}} p(x_T</td>
</tr>
</tbody>
</table>
Table 7.2: List of publications for each chapter

<table>
<thead>
<tr>
<th>Category</th>
<th>Publications</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LWF CGs</strong></td>
<td>• Finding Minimal Separators in LWF Chain Graphs (Javidian and Valtorta, 2018b) [section 2.3].</td>
</tr>
<tr>
<td></td>
<td>• Learning LWF Chain Graphs: An Order-Independent Algorithm (Javidian, Valtorta, and Jamshidi, 2019b) [section 2.4].</td>
</tr>
<tr>
<td><strong>MVR CGs</strong></td>
<td>• On the properties of MVR Chain Graphs (Javidian and Valtorta, 2018c) [section 3.2].</td>
</tr>
<tr>
<td></td>
<td>• Finding Minimal Separators in MVR Chain Graphs (Javidian and Valtorta, 2018a) [section 3.3].</td>
</tr>
<tr>
<td></td>
<td>• Order-Independent Structure Learning of Multivariate Regression Chain Graphs (Javidian, Valtorta, and Jamshidi, 2019c) [section 3.4]</td>
</tr>
<tr>
<td></td>
<td>• A Decomposition-Based Algorithm for Learning the Structure of MVR Chain Graphs (Javidian and Valtorta, 2019a) [section 3.5].</td>
</tr>
<tr>
<td><strong>AMP CGs</strong></td>
<td>• AMP CGs: Minimal Separators and Structure Learning Algorithms (Javidian, Valtorta, and Jamshidi, 2019a) [Chapter 4].</td>
</tr>
<tr>
<td><strong>Bayesian hypergraphs</strong></td>
<td>• On a hypergraph probabilistic graphical model (Javidian et al., 2018; Wang et al., 2019) [Chapter 5].</td>
</tr>
<tr>
<td><strong>Causal transfer learning</strong></td>
<td>• Transfer Learning for Performance Modeling of Configurable Systems: A Causal Analysis (Javidian, Jamshidi, and Valtorta, 2019) [Chapter 6].</td>
</tr>
</tbody>
</table>


Javidian, M. A.; Valtorta, M.; and Jamshidi, P. 2019a. AMP CGs: Minimal separators and structure learning algorithms. Submitted to JMLR.


Richardson, T. S. 2018. Personal communication.


Appendix A

Proofs of Correctness of the Theorems and Algorithms in Section 2.4

Before proving the correctness of Algorithm 4, we need several lemmas.

Lemma 1. After line 10 of Algorithm 4, $G$ and $H$ have the same adjacencies.

Proof. Consider any pair of nodes $A$ and $B$ in $G$. If $A \in \text{ad}_G(B)$, then $A \not\perp B|S$ for all $S \subseteq V \setminus (A \cup B)$ by the faithfulness assumption. Consequently, $A \in \text{ad}_H(B)$ at all times. On the other hand, if $A \notin \text{ad}_G(B)$ (equivalently $B \notin \text{ad}_G(A)$), Algorithm 2 (Javidian and Valtorta, 2018b) returns a set $Z \subseteq \text{ad}_H(A) \setminus B$ (or $Z \subseteq \text{ad}_H(B) \setminus A$) such that $A \not\perp p_B Z$. This means there exist $0 \leq i \leq |V_H| - 2$ such that the edge $A - B$ is removed from $H$ in line 7. Consequently, $A \notin \text{ad}_H(B)$ after line 10. □

Lemma 2. $G$ and $H^*$ have the same minimal complexes and adjacencies after line 19 of Algorithm 4.

Proof. $G$ and $H^*$ have the same adjacencies by Lemma 1. Now we show that any arrow that belongs to a minimal complex in $G$ is correctly oriented in line 15 of Algorithm 4, in the sense that it is an arrow with the same orientation in $G$. For this purpose, consider the following two cases:

Case 1: $u \to w \leftarrow v$ is an induced subgraph in $G$. So, $u, v$ are not adjacent in $H$ (by Lemma 1), $u - w \in H^*$ (by Lemma 1), and $u \not\perp_p v|(S_{uv} \cup \{w\})$ by the faithfulness assumption. So, $u - w$ is oriented as $u \to w$ in $H^*$ in line 15. Obviously, we will not orient it as $w \to u$.  

239
Case 2: $u \rightarrow w \cdots z \leftarrow v$, where $w \neq z$ is a minimal complex in $G$. So, $u, v$ are not adjacent in $H$ (by Lemma 1), $u - w \in H^*$ (by Lemma 1), and $u \perp_p v|_w (S_{wv} \cup \{w\})$ by the faithfulness assumption. So, $u - w$ is oriented as $u \rightarrow w$ in $H^*$ in line 15. Since $u \in S_{vw}$ and $w \perp_p v|_w (S_{wv} \cup \{u\})$ by the faithfulness assumption so $u, v$, and $w$ do not satisfy the conditions and hence we will not orient $u - w$ as $w \rightarrow u$.

![Diagram](image_url)

Figure A.1: (a) The LWF CG $G$, (b) the skeleton of $G$, (c) $H^*$ before executing line 19 in Algorithm 4, and (d) $H^*$ after executing line 19 in Algorithm 4.

Consider the chain graph $G$ in Figure A.1(a). After applying the skeleton recovery of Algorithm 4, we obtain $H$, the skeleton of $G$, in Figure A.1(b). In the execution of the complex recovery of Algorithm 4, when we pick $A, B$ in line 12 and $C$ in line 13, we have $A \perp B|\emptyset$, that is, $S_{AB} = \emptyset$, and find that $A \perp B|C$. Hence we orient $B - C$ as $B \rightarrow C$ in line 15, which is not a complex arrow in $G$. Note that we do not orient $C - B$ as $C \rightarrow B$: the only chance we might do so is when $u = C, v = A$, and $w = B$ in the inner loop of the complex recovery of Algorithm 4, but we have $B \in S_{AC}$ and the condition in line 14 is not satisfied. Hence, the graph we obtain before the last step of complex recovery in Algorithm 4 must be the one given in Figure A.1(c), which differs from the recovered pattern in Figure A.1(d). This illustrates the necessity of the last step of complex recovery in Algorithm 4. To see how the edge $B \rightarrow C$ is removed in the last step of complex recovery in Algorithm 4, we observe that, if we follow the procedure described in the comment after line 19 of Algorithm 4, the only chance that $B \rightarrow C$ becomes one of the candidate complex arrow pair is when it is considered together with $A \rightarrow D$. However, the only undirected path
between $C$ and $D$ is simply $C - D$ with $D$ adjacent to $B$. Hence $B \rightarrow C$ stays unlabeled and will finally get removed in the last step of complex recovery in Algorithm 4.

Consequently, $G$ and $H^*$ have the same minimal complexes and adjacencies after line 19. □

Proof of Theorem 2.8. The proof of Theorem 2.8 is completely analogous to the proof of the correctness of the original PC-like algorithm. □

Proof of Theorem 2.9. We consider the removal or retention of an arbitrary edge $u \rightarrow v$ at some level $i$. The ordering of the variables determines the order in which the edges (line 7 of Algorithm 5) and the subsets $S$ of $a_H(u)$ and $a_H(v)$ (line 8 of Algorithm 5) are considered. By construction, however, the order in which edges are considered does not affect the sets $a_H(u)$ and $a_H(v)$.

If there is at least one subset $S$ of $a_H(u)$ or $a_H(v)$ such that $u \perp \perp P v \mid S$, then any ordering of the variables will find a separating set for $u$ and $v$. (Different orderings may lead to different separating sets as illustrated in Example 2, but all edges that have a separating set will eventually be removed, regardless of the ordering). Conversely, if there is no subset $S'$ of $a_H(u)$ or $a_H(v)$ such that $u \perp \perp P v \mid S'$, then no ordering will find a separating set.

Hence, any ordering of the variables leads to the same edge deletions, and therefore to the same skeleton. □

Proof of Theorem 2.10. The skeleton of the learned pattern is correct by Theorem 2.8. Since $u, v$ are not adjacent they are $c$-separated given some subset $S \setminus \{u, v\}$ (see Algorithm 2). Based on the $c$-separation criterion for LWF CGs (see section 2), if $w$ is a node on a minimal complex in $G$ such that $u$ and $w$ are adjacent then $u \perp \perp P v \mid S \cup \{w\}$ for any $S \setminus \{u, v\}$ due to the moralization procedure. As a result, $u - w$ edges are all unambiguous and so the $U$-structures are correct as in the CPC/MPC-like algorithm. Therefore, the output of the (stable) CPC/MPC-like algorithm is a pattern that is Markov equivalent to $G$. □
Proof of Theorem 2.11. The stable CPC/MPC-like algorithm have order-independent skeleton, by Theorem 2.9. In particular, this means that their adjacency sets are order-independent. For non adjacent nodes $u$ and $v$ the decision about whether the undirected edge $u - w$ is unambiguous and/or a $U$-structure is based on the adjacency sets of nodes $u$ and $v$, which are order independent. The rest of theorem follows straightforwardly from Theorems 2.9 and the first part of this proof. □
APPENDIX B

PROOF OF THEOREM 3.10

Proof. (i)⇒(ii): This has already been proved in (Richardson, 2003, Theorem 1).

(ii)⇒(iii): Assume that the independence model $\mathcal{F}$ over the node set of MVR CG($G$) satisfies the global Markov property w.r.t. $G$ in Definition 3.3. We have the following three cases:

Case 1: Let $X = \tau \in \mathcal{T}, Y = nd_D(\tau) \setminus pa_D(\tau)$, and $Z = pa_D(\tau)$. So, $an(X \cup Y \cup Z) = \tau \cup nd_D(\tau)$ is an ancestor set, and $pa_D(\tau)$ separates $\tau$ from $nd_D(\tau) \setminus pa_D(\tau)$ in $(G_{\tau \cup nd_D(\tau)})^a$; this shows that the global Markov property in Definition 3.3 implies (IV0) in Definition 3.8.

Case 2: Assume that $X = \sigma \subseteq \tau \in \mathcal{T}, Y = pa_D(\tau) \setminus pa_G(\sigma)$, and $Z = pa_G(\sigma)$. Consider that $W = an(X \cup Y \cup Z) = an(\sigma \cup pa_D(\tau))$. We know that there is no directed edge from $pa_D(\tau) \setminus pa_G(\sigma)$ to elements of $\sigma$, and also there is no collider path between nodes of $Y$ and $\sigma$ in $W$. So, every connecting path that connects $pa_D(\tau) \setminus pa_G(\sigma)$ to $\sigma$ in $(G_W)^a$ has intersection with $pa_G(\sigma)$, which means $pa_G(\sigma)$ separates $pa_D(\tau) \setminus pa_G(\sigma)$ from $\sigma$ in $(G_W)^a$; this shows that the global Markov property in Definition 3.3 implies (IV1) in Definition 3.8.

Case 3: Assume that $X = \sigma \subsetneq \tau \in \mathcal{T}$ is a connected subset of $\tau$. Also, assume that $Y = \tau \setminus Nb_G(\sigma)$, and $Z = pa_D(\tau)$. Obviously, $\sigma$ and $\tau \setminus Nb_G(\sigma)$ are two subsets of $\tau$ such that there is no connection between their elements. Consider that $A$ is the ancestor set containing $\sigma$, $\tau \setminus Nb_G(\sigma)$, and $pa_D(\tau)$. Clearly, $pa_D(\tau) \subseteq A$. Since $\sigma$ and $\tau \setminus Nb_G(\sigma)$ are disconnected in $\tau$, so any connecting path between them in $A$ (if it exists) must pass through $pa_D(\tau)$ in $(G_A)^a$; this shows that the global Markov property in Definition 3.3 implies (IV2) in Definition 3.8.

(iii)⇒(iv): Assume that the independence model $\mathcal{F}$ over the node set of MVR CG($G$)
satisfies the block recursive Markov property w.r.t. \( G \) in Definition 3.8. We show that \( \mathcal{I} \) satisfies the MR Markov property w.r.t. \( G \) in Definition 3.7 by considering the following two cases:

**Case 1 (IV0 and IV1 ⇒ MR1):** Assume that \( A \) is a connected subset of \( \tau \). From (IV1) we have:

\[
A \perp (\text{pa}_D(\tau) \setminus \text{pa}_G(A))|\text{pa}_G(A) \tag{B.1}
\]

Also, from (IV0) we have \( \tau \perp (\text{nd}_D(\tau) \setminus \text{pa}_G(A))|\text{pa}_D(\tau) \), the decomposition property implies that

\[
A \perp (\text{nd}_D(\tau) \setminus \text{pa}_G(A))|\text{pa}_D(\tau) \tag{B.2}
\]

Using the contraction property for (B.1) and (B.2) gives:

\[
A \perp [(\text{nd}_D(\tau) \setminus \text{pa}_G(A)) \cup (\text{pa}_D(\tau) \setminus \text{pa}_G(A))]|\text{pa}_G(\tau)
\]

Using the decomposition property for this independence relationship gives (MR1): \( A \perp (\text{pre}(\tau) \setminus \text{pa}_G(A))|\text{pa}_G(A) \), because \( (\text{pre}(\tau) \setminus \text{pa}_G(A)) \subseteq [(\text{nd}_D(\tau) \setminus \text{pa}_G(A)) \cup (\text{pa}_D(\tau) \setminus \text{pa}_G(A))] \).

**Case 2 (IV0 and IV2 ⇒ MR2):** Consider that \( A \) is a disconnected subset of \( \tau \) that contains \( r \) connected components \( A_1, \ldots, A_r \) i.e., \( A = A_1 \cup \cdots \cup A_r \). From (IV2) we have: \( A_1 \perp \tau \setminus \text{Nb}_G(A_1)|\text{pa}_D(\tau) \). Using the decomposition property gives:

\[
A_1 \perp A_2|\text{pa}_D(\tau) \tag{B.3}
\]

Also, using decomposition for (IV0) gives: \( (A_1 \cup A_2) \perp (\text{pre}(\tau) \setminus \text{pa}_D(\tau))|\text{pa}_D(\tau) \). Applying the weak union property for this independence relation gives: \( A_1 \perp (\text{pre}(\tau) \setminus \text{pa}_D(\tau))|A_2 \cup \text{pa}_D(\tau) \). Using the contraction property for this and (B.3) gives: \( A_1 \perp [A_2 \cup (\text{pre}(\tau) \setminus \text{pa}_D(\tau))]|\text{pa}_D(\tau) \). Using the weak union property leads to \( A_1 \perp A_2|(\text{pa}_D(\tau) \cup (\text{pre}(\tau) \setminus \text{pa}_D(\tau))) = \text{pre}(\tau) \). Similarly, we can prove that for every \( 1 \leq i \neq j \leq r \): \( A_i \perp A_j|\text{pre}(\tau) \).

**(iv)⇒(v):** Assume that the independence model \( \mathcal{J} \) over the node set of MVR CG(\( G \)) satisfies the MR Markov property w.r.t. \( G \) in Definition 3.7, and \( \prec \) is an ordering that is consistent with \( G \). Let \( x \in A \subseteq \text{pre}_{G,\prec}(x) \), We show that \( \mathcal{J} \) satisfies the ordered local Markov property w.r.t. \( G \) in Definition 3.9 by considering the following two cases:
Case 1: There is a chain component $T$ such that $x \in T$. Consider that $A \cap T$ is a connected subset of $T$. From (MR1) we have: $\text{dis}_{G_A}(x) \perp \text{pre}(T) \setminus \text{pa}_G(\text{dis}_{G_A}(x))|\text{pa}_G(\text{dis}_{G_A}(x))$. Using the weak union property gives: $\{x\} \perp \text{pre}(T) \setminus \text{pa}_G(\text{dis}_{G_A}(x))|\text{pa}_G(\text{dis}_{G_A}(x)) \cup (\text{dis}_{G_A}(x) \setminus \{x\})$. Since $[A \setminus (\text{mb}(x,A) \cup \{x\})] \subseteq [\text{pre}(T) \setminus \text{pa}_G(\text{dis}_{G_A}(x))]$, using the decomposition property leads to: $\{x\} \perp [A \setminus (\text{mb}(x,A) \cup \{x\})]|\text{mb}(x,A)$.

Case 2: There is a chain component $T$ such that $x \in T$, and $A \cap T$ is a disconnected subset of $T$ with connected components $A_1, \ldots, A_k$ i.e., $A \cap T = A_1 \cup \cdots \cup A_k$. It is clear that there is a $1 \leq d \leq k$ such that $A_d = \text{dis}_{G_A}(x)$. We have the following two sub-cases:

Sub-case I): $\sigma := T \setminus \text{Nb}_G(A_d)$ is a connected subset of $T$.

\[
\begin{align*}
\text{From (MR2): } A_d & \perp \sigma|\text{pre}(T) \\
\text{From (MR1): } A_d & \perp (\text{pre}(T) \setminus \text{pa}_G(A_d))|\text{pa}_G(A_d)
\end{align*}
\]  

Using the contraction property for (B.4) gives: $A_d \perp [\sigma \cup (\text{pre}(T) \setminus \text{pa}_G(A_d))]|\text{pa}_G(A_d)$.

Using the weak union property gives: $\{x\} \perp [\text{pre}(T) \setminus \text{pa}_G(\text{dis}_{G_A}(x))]|\text{pa}_G(\text{dis}_{G_A}(x)) \cup (\text{dis}_{G_A}(x) \setminus \{x\})$. Since $[A \setminus (\text{mb}(x,A) \cup \{x\})] \subseteq [\text{pre}(T) \setminus \text{pa}_G(\text{dis}_{G_A}(x))]$, using the decomposition property leads to: $\{x\} \perp [A \setminus (\text{mb}(x,A) \cup \{x\})]|\text{mb}(x,A)$.

Sub-case II): $T \setminus \text{Nb}_G(A_d)$ is a disconnected subset of $T$ with connected component $\sigma_1, \sigma_2$ i.e., $T \setminus \text{Nb}_G(A_d) = \sigma_1 \cup \sigma_2$. From (MR1) we have: $\sigma_1 \perp (T \setminus \text{Nb}_G(\sigma_1))|\text{pre}(T)$. Since $(A_d \cup \sigma_2) \subseteq (T \setminus \text{Nb}_G(\sigma_1))$, using the decomposition and weak union property give: $\sigma_1 \perp A_d|(\text{pre}(T) \cup \sigma_2)$. Using the symmetry property implies that $A_d \perp \sigma_1|(\text{pre}(T) \cup \sigma_2)$.

\[
\begin{align*}
A_d & \perp \sigma_1|(\text{pre}(T) \cup \sigma_2) \\
\text{From (MR2): } A_d & \perp \sigma_2|\text{pre}(T)
\end{align*}
\]  

Using the contraction property for (B.5) gives: $A_d \perp (\sigma_1 \cup \sigma_2)|\text{pre}(T)$.

\[
\begin{align*}
A_d & \perp (\sigma_1 \cup \sigma_2)|\text{pre}(T) \\
\text{From (MR1): } A_d & \perp (\text{pre}(T) \setminus \text{pa}_G(A_d))|\text{pa}_G(A_d)
\end{align*}
\]  

Using the contraction property for (B.6) gives:

$A_d \perp [(\sigma_1 \cup \sigma_2) \cup (\text{pre}(T) \setminus \text{pa}_G(A_d))]|\text{pa}_G(A_d)$

245
Using the decomposition property gives: \([x] \perp \perp [(\sigma_1 \cup \sigma_2) \cup (pre(T) \setminus pa_G(A_d))]mb(x, A)\).

Since \([A \setminus (mb(x, A) \cup \{x\})] \subseteq [(\sigma_1 \cup \sigma_2) \cup (pre(T) \setminus pa_G(A_d))]\), using the decomposition property leads to: \([x] \perp \perp [A \setminus (mb(x, A) \cup \{x\})]mb(x, A)\).

\((v)\Rightarrow(i)\): This has already been proved in (Richardson, 2003, Theorem 2). \(\square\)
Appendix C

Proofs of Theoretical Results in Section 3.5

Lemma 3. Let $\rho$ be a chain from $u$ to $v$, and $W$ be the set of all vertices on $\rho$ ($W$ may or may not contain $u$ and $v$). Suppose that (the endpoints of) a chain $\rho$ is blocked by $S$. If $W \subseteq S$, then the chain $\rho$ is blocked by $W$ and by any set containing $W$.

Proof. Since the blocking of the chain $\rho$ depends on those vertices between $u$ and $v$ that are contained in the $m$-separator, and since $W$ contains all vertices on $\rho$, $\rho$ is also blocked by $S \cap W = W$ if $\rho$ is blocked by $S$. Since all colliders on $\rho$ have already been activated conditionally on $W$, adding other vertices into the conditional set does not make any new collider active on $\rho$. This implies that $\rho$ is blocked by any set containing $W$. □

Lemma 4. Let $T$ be an $m$-separation tree for CG $G$, and $K$ be a separator of $T$ that separates $T$ into two subtrees $T_1$ and $T_2$ with variable sets $V_1$ and $V_2$ respectively. Suppose that $\rho$ is a chain from $u$ to $v$ in $G$ where $u \in V_1 \setminus K$ and $v \in V_2 \setminus K$. Let $W$ denote the set of all vertices on $\rho$ ($W$ may or may not contain $u$ and $v$). Then the chain $\rho$ is blocked by $W \cap K$ and by any set containing $W \cap K$.

Proof. Since $u \in V_1 \setminus K$ and $v \in V_2 \setminus K$, there is a sequence from $s$ (may be $u$) to $y$ (may be $v$) in $\rho = (u, \ldots, s, t, \ldots, x, y, \ldots, v)$ such that $s \in V_1 \setminus K$ and $y \in V_2 \setminus K$ and all vertices from $t$ to $x$ are contained in $K$. Let $\rho'$ be the sub-chain of $\rho$ from $s$ to $y$ and $W'$ the vertex set from $t$ to $x$, so $W' \subseteq K$. Since $s \in V_1 \setminus K$ and $y \in V_2 \setminus K$, we have from definition of $m$-separation tree that $K$ $m$-separates $s$ and $y$ in $G$, i.e., $K$ blocks $\rho'$. By lemma 3, we obtain that $\rho'$ is blocked by $W'(\subseteq K)$ and any set containing $W'$. Since $W' \subseteq (K \cap W)$, $\rho'$...
is blocked by $K \cap W$ and by any set containing $K \cap W$. Thus $\rho(\supseteq \rho')$ is also blocked by them. □

**Remark 15.** Javidian and Valtorta showed that if we find a separator over $S$ in $(G_{An(u \cup v)})^a$ then it is an m-separator in $G$. On the other hand, if there exists an m-separator over $S$ in $G$ then there must exist a separator over $S$ in $(G_{An(u \cup v)})^a$ by removing all nodes which are not in $An(u \cup v)$ from it (Javidian and Valtorta, 2018a).

Observations in Remark 15 yield the following results.

**Lemma 5.** Let $u$ and $v$ be two non-adjacent vertices in MVR CG $G$, and let $\rho$ be a chain from $u$ to $v$. If $\rho$ is not contained in $An(u \cup v)$, then $\rho$ is blocked by any subset $S$ of $an(u \cup v)$.

**Proof.** Since $\rho \not\subseteq An(u \cup v)$, there is a sequence from $s$ (may be $u$) to $y$ (may be $v$) in $\rho = (u, \ldots, s, t, \ldots, x, y, \ldots, v)$ such that $s$ and $y$ are contained in $An(u \cup v)$ and all vertices from $t$ to $x$ are out of $An(u \cup v)$. Then the edges $s - t$ and $x - y$ must be oriented as $s \rightsquigarrow t$ and $x \leftarrow y$, otherwise $t$ or $x$ belongs to $an(u \cup v)$. Thus there exist at least one collider between $s$ and $y$ on $\rho$. The middle vertex $w$ of the collider closest to $s$ between $s$ and $y$ is not contained in $an(u \cup v)$, and any descendant of $w$ is not in $an(u \cup v)$, otherwise there is a (partially) directed cycle. So $\rho$ is blocked by the collider, and it cannot be activated conditionally on any vertex in $S$ where $S \subseteq an(u \cup v)$. □

**Lemma 6.** Let $T$ be an m-separation tree for CG $G$. For any vertex $u$ there exists at least one node of $T$ that contains $u$ and $bd(u)$.

**Proof.** If $bd(u)$ is empty, it is trivial. Otherwise let $C$ denote the node of $T$ which contains $u$ and the most number of elements of $u$’s boundary. Since no set can separate $u$ from a parent (or neighbor), there must be a node of $T$ that contains $u$ and the parent (or neighbor). If $u$ has only one parent (or neighbor), then we obtain the lemma. If $u$ has two or more elements in its boundary, we choose two arbitrary elements $v$ and $w$ of $u$’s boundary that are not contained in a single node but are contained in two different nodes of $T$, say $\{u, v\} \subseteq C$ and
\{u, w\} \subseteq C\prime$ respectively, since all vertices in $V$ appear in $T$. On the chain from $C$ to $C\prime$ in $T$, all separators must contain $u$, otherwise they cannot separate $C$ from $C\prime$. However, any separator containing $u$ cannot separate $v$ and $w$ because $v \leftrightarrow u \leftrightarrow w$ is an active chain between $v$ and $w$ in $G$. Thus we got a contradiction. \hfill \Box

**Lemma 7.** Let $T$ be an $m$-separation tree for $CG\ G$ and $C$ a node of $T$. If $u$ and $v$ are two vertices in $C$ that are non-adjacent in $G$, then there exists a node $C\prime$ of $T$ containing $u, v$ and a set $S$ such that $S$ $m$-separates $u$ and $v$ in $G$.

**Proof.** Without loss of generality, we can suppose that $v$ is not a descendant of the vertex $u$ in $G$, i.e., $v \notin nd(u)$. According to the local Markov property for MVR chain graphs proposed by Javidian and Valtorta in (Javidian and Valtorta, 2018c), we know that $u \perp \perp nd(u) \setminus bd(u) \left| \text{pa}_G(u)$. By Lemma 6, there is a node $C_1$ of $T$ that contains $u$ and $bd(u)$. If $v \in C_1$, then $S$ defined as the parents of $u$ $m$-separates $u$ from $v$.

If $v \notin C_1$, choose the node $C_2$ that is the closest node in $T$ to the node $C_1$ and that contains $u$ and $v$. Consider that there is at least one parent (or neighbor) $p$ of $u$ that is not contained in $C_2$. Thus there is a separator $K$ connecting $C_2$ toward $C_1$ in $T$ such that $K$ $m$-separates $p$ from all vertices in $C_2 \setminus K$. Note that on the chain from $C_1$ to $C_2$ in $T$, all separators must contain $u$, otherwise they cannot separate $C_1$ from $C_2$. So, we have $u \in K$ but $v \notin K$ (if $v \in K$, then $C_2$ is not the closest node of $T$ to the node $C_1$). In fact, for every parent (or neighbor) $p'$ of $u$ that is contained in $C_1$ but not in $C_2$, $K$ separates $p'$ from all vertices in $C_2 \setminus K$, especially the vertex $v$.

Define $S = (an(u \cup v) \cap C_2)$, which is a subset of $C_2$. We need to show that $u$ and $v$ are $m$-separated by $S$, that is, every chain between $u$ and $v$ in $G$ is blocked by $S$.

If $\rho$ is not contained in $An(u \cup v)$, then we obtain from Lemma 5 that $\rho$ is blocked by $S$.

When $\rho$ is contained in $An(u \cup v)$, let $x$ be adjacent to $u$ on $\rho$, that is, $\rho = (u, x, y, \ldots, v)$. We consider the three possible orientations of the edge between $u$ and $x$. We now show that $\rho$ is blocked in all three cases.
i: \( u \leftarrow x \), so we know that \( x \) is not a collider and we have two possible sub-cases:

1. \( x \in C_2 \). In this case the chain \( \rho \) is blocked at \( x \).
2. \( x \not\in C_2 \). In this case \( K m \)-separates \( x \) from \( v \). By Lemma 4, we can obtain that the sub-chain \( \rho' \) from \( x \) to \( v \) can be blocked by \( W \cap K \) where \( W \) denotes the set of all vertices between \( x \) and \( v \) (not containing \( x \) and \( v \)) on \( \rho' \). Since \( S \supseteq (W \cap K) \), we obtain from Lemma 4 that \( S \) also blocks \( \rho' \). Hence the chain \( \rho \) is blocked by \( S \).

ii: \( u \rightarrow x \). We have the following sub-cases:

1. \( x \in an(u) \). This case is impossible because a directed cycle would occur.
2. \( x \in an(v) \). This case is impossible because \( v \) cannot be a descendant of \( u \).

iii: \( u \leftrightarrow x \). We have the following sub-cases:

1. \( x \in an(u) \). This case is impossible because a partially directed cycle would occur.
2. \( x \in an(v) \) and \( v \) is in the same chain component \( \tau \) that contains \( u, x \). This is impossible, because in this case we have a partially directed cycle.
3. \( x \in an(v) \) and \( v \) is not in the same chain component \( \tau \) that contains \( u, x \). We have the following sub-cases:

   - \( x \not\in C_2 \). In this case \( K m \)-separates \( x \) from \( v \). By Lemma 4, we can obtain that the sub-chain \( \rho' \) from \( x \) to \( v \) can be blocked by \( W \cap K \) where \( W \) denotes the set of all vertices between \( x \) and \( v \) (not containing \( x \) and \( v \)) on \( \rho' \). Since \( S \supseteq (W \cap K) \), we obtain from Lemma 4 that \( S \) also blocks \( \rho' \). Hence the chain \( \rho \) is blocked by \( S \).
   - \( x \in C_2 \). We have the three following sub-cases:
     * \( u \leftrightarrow x \rightarrow y \). In this case \( x \in S \) blocks the chain. Note that in this case it is possible that \( y = v \).
\* \( u \leftrightarrow x \leftarrow y \). So, \( y \neq v \text{ o.w., a directed cycle would occur} \) is not a collider. If \( y \in C_2 \) then the chain \( \rho \) is blocked at \( y \). Otherwise, we have the two following sub-cases:

- There is a node \( C' \) between \( C_1 \) and \( C_2 \) that contains \( y \) (note that it is possible that \( C' = C_1 \)), so \( K \) \( m \)-separates \( y \) from \( v \) and the same argument used for case i.2 holds.
- In this case \( K \) \( m \)-separates \( y \) from \( p \) (\( p \in \text{bd}(u) \cap C_1 \) and \( p \notin C_2 \)), which is impossible because the chain \( p \leftarrow u \leftrightarrow x \leftarrow y \) is active (note that \( u, x \in K \)).

\* \( u \leftrightarrow x \leftarrow y \). If there is an outgoing (\( \rightarrow \)) edge from \( y \) (\( \neq v \text{ o.w., a partially directed cycle would occur} \)) then the same argument in the previous sub-case \( (u \leftrightarrow x \leftarrow y) \) holds. Otherwise, \( y \) is a collider. If \( y \notin C_2 \) then the chain \( \rho \) is blocked at \( y \). If \( y \in C_2 \), there must be a non-collider vertex on the chain \( \rho \) between \( y \) and \( v \) to prevent a (partially) directed cycle. The same argument as in the previous sub-case \( (u \leftrightarrow x \leftarrow y) \) holds.

\( \square \)

Proof of Theorem 3.33. From (Cowell et al., 1999), we know that any separator \( S \) in junction tree \( T \) separates \( V_1 \setminus S \) and \( V_2 \setminus S \) in the triangulated graph \( \tilde{G}_V^i \), where \( V_i \) denotes the variable set of the subtree \( T_i \) induced by removing the edge with a separator \( S \) attached, for \( i = 1, 2 \). Since the edge set of \( \tilde{G}_V^i \) contains that of undirected independence graph \( \tilde{G}_V \) for \( G \), \( V_1 \setminus S \) and \( V_2 \setminus S \) are also separated in \( \tilde{G}_V \). Since \( \tilde{G}_V \) is an undirected independence graph for \( G \), using Definition 3.32 we obtain that \( T \) is an \( m \)-separation tree for \( G \). \( \square \)

Proof of Theorem 3.34. (\( \Rightarrow \)) If condition (i) is the case, nothing remains to prove. Otherwise, Lemma 7 implies condition (ii).
(⇐) Assume that \(u\) and \(v\) are not contained together in any node \(C\) of \(T\). Also, assume that \(C_1\) and \(C_2\) are two nodes of \(T\) that contain \(u\) and \(v\), respectively. Consider that \(C'_1\) is the most distant node from \(C_1\), between \(C_1\) and \(C_2\), that contains \(u\) and \(C'_2\) is the most distant node from \(C_2\), between \(C_1\) and \(C_2\), that contains \(v\). Note that it is possible that \(C'_1 = C_1\) or \(C'_2 = C_2\). By the condition (i) we know that \(C'_1 \neq C'_2\). Any separator between \(C'_1\) and \(C'_2\) satisfies the assumptions of Lemma 4. The sufficiency of condition (i) is given by Lemma 4.

The sufficiency of conditions (ii) is trivial by the definition of \(m\)-separation. \(\square\)
APPENDIX D

PROOFS OF CORRECTNESS OF ALGORITHMS 11 AND 12

Correctness of Algorithm 11. Since an augmented graph for CG $G$ is an undirected independence graph, by definition of an undirected independence graph, it is enough to show that $\tilde{G}_V$ defined in step 3 contains all edges of $(G_V)^a$. It is obvious that $\tilde{E}$ contains all edges obtained by dropping directions of directed edges in $G$ since any set cannot $m$-separate two vertices that are adjacent in $G$.

Now we show that $\tilde{E}$ also contains any augmented edge that connects vertices $u$ and $v$ having a collider chain between them, that is, $(u, v) \in \tilde{E}$. Any chain graph yields a directed acyclic graph $D$ of its chain components having $T$ as a node set and an edge $T_1 \rightarrow T_2$ whenever there exists in the chain graph $G$ at least one edge $u \rightarrow v$ connecting a node $u$ in $T_1$ with a node $v$ in $T_2$ (Marchetti and Lupparelli, 2011). So, there is a collider chain between two nodes $u$ and $v$ if and only if there is a chain component $\tau \in T$ such that

1. $u, v \in \tau$, or
2. $u \in \tau$ and $v \in pa_G(\tau)$ or vice versa, or
3. $u, v \in pa_G(\tau)$

Since for each connected component $\tau$ there is a $C_h \in C$ containing both $\tau$ and its parent set $pa_G(\tau)$, in all of above mentioned cases we have an $(u, v)$ edge in step 2. Therefore, $\tilde{G}_V$ defined in step 3 contains all edges of $(G_V)^a$. □

Correctness of Algorithm 12. By the sufficiency of Theorem 3.34, the initializations at steps 2 and 3 for creating edges guarantee that no edge is created between any two vari-
ables which are not in the same node of the $m$-separation tree. Also, by the sufficiency of Theorem 3.34, deleting edges at steps 2 and 3 guarantees that any other edge between two $m$-separated variables can be deleted in some local skeleton. Thus the global skeleton obtained at step 3 is correct. In a maximal ancestral graph, every missing edge corresponds to at least one independency in the corresponding independence model (Richardson and Spirtes, 2002), and MVR CGs are a subclass of maximal ancestral graphs (Javidian and Valtorta, 2018c). Therefore, according to the necessity of Theorem 3.34, each augmented edge $(u, v)$ in the undirected independence graph must be deleted at some subgraph over a node of the $m$-separation tree. Furthermore, according to Lemma 6, for every $v$-structure $(u \leftrightarrow w \leftrightarrow v)$ there is a node in $m$-separation tree $T$ that contains $u, v$ and $w$, and obviously $w \notin S_{uv}$. Therefore, we can determine all $v$-structures at step 4, which completes our proof. □
APPENDIX E

PROOFS OF CORRECTNESS OF THE ALGORITHMS IN SECTIONS 4.4 AND 4.5

In Theorem 4.8, we showed that if we find a separator over $S$ in $(G_{\text{ant}(u \cup v)})^a$ then it is a $p$-separator in $G$. On the other hand, if there exists a $p$-separator over $S$ in $G$ then there must exist a separator over $S$ in $(G_{\text{ant}(u \cup v)})^a$ by removing all nodes which are not in $\text{ant}(u \cup v)$ from it. This observation yields the following results.

Lemma 8. Let $u$ and $v$ be two non-adjacent vertices in AMP CG $G$, and let $\rho$ be a chain from $u$ to $v$. If $\rho$ is not contained in $\text{ant}(u \cup v)$, then $\rho$ is blocked by any subset $S$ of $\text{ant}(u \cup v) \setminus \{u, v\}$.

Proof. Since $\rho \not\subseteq \text{ant}(u \cup v)$, there is a sequence from $s$ (may be $u$) to $y$ (may be $v$) in $\rho = (u, \ldots, s, t, \ldots, x, y, \ldots, v)$ such that $s$ and $y$ are contained in $\text{ant}(u \cup v)$ and all vertices from $t$ to $x$ are out of $\text{ant}(u \cup v)$. Then the edges $s \rightarrow t$ and $x \leftarrow y$ must be oriented as $s \rightarrow t$ and $x \leftarrow y$, otherwise $t$ or $x$ belongs to $\text{ant}(u \cup v)$. Thus there exist at least one triplex between $s$ and $y$ on $\rho$. The middle vertex $w$ of the triplex closest to $s$ between $s$ and $y$ is not contained in $\text{ant}(u \cup v)$, and any descendant of $w$ is not in $\text{ant}(u \cup v)$. So $\rho$ is blocked by this triplex, and it cannot be activated conditionally on any vertex in $S$ where $S \subseteq \text{ant}(u \cup v) \setminus \{u, v\}$.

Lemma 9. Let $T$ be a $p$-separation tree for the AMP CG $G$. For any vertex $u$ there exists at least one node of $T$ that contains $u$ and $\text{pa}(u)$.

Proof. If $\text{pa}(u)$ is empty, the result is trivial. Otherwise let $C$ denote the node of $T$ which contains $u$ and the most elements of $u$’s parent. Since no set can separate $u$ from a parent,
there must be a node of $T$ that contains $u$ and the parent. If $u$ has only one parent, then we
obtain the lemma. If $u$ has two or more parents, we choose two arbitrary elements $v$ and $w$
of $u$’s parent that are not contained in a single node of $T$ but are contained in two different
nodes of $T$, say $\{u, v\} \subseteq C$ and $\{u, w\} \subseteq C'$ respectively, since all vertices in $V$ appear in
$T$. On the chain from $C$ to $C'$ in $T$, all separators must contain $u$, otherwise they cannot
separate $C$ from $C'$. However, any separator containing $u$ cannot separate $v$ and $w$ because
$v \rightarrow u \leftarrow w$ is an active triplex between $v$ and $w$ in $G$. Thus we got a contradiction. □

Lemma 10. Let $T$ be a $p$-separation tree for AMP CG $G$ and $C$ a node of $T$. If $u$ and $v$ are
two vertices in $C$ that are non-adjacent in $G$ and belong to two different chain components,
then there exists a node $C'$ of $T$ containing $u, v$ and a set $S$ such that $S$ $p$-separates $u$ and
$v$ in $G$.

Proof. Assume that $u$ and $v$ are two vertices in $G$ that are non-adjacent and belong to two
different chain components. Without loss of generality, we can suppose that $v$ is not a
descendant of the vertex $u$ in $G$, i.e., $v \notin nd(u)$. According to the pairwise Markov property
for AMP chain graphs in (Andersson, Madigan, and Perlman, 2001), $u \perp \perp v|pa(u)$. By
Lemma 9, there is a node $C_1$ of $T$ that contains $u$ and $pa(u)$. If $v \in C_1$, then $S$ defined as
the parents of $u$ $p$-separates $u$ from $v$.

If $v \notin C_1$, choose the node $C_2$ that is the closest node in $T$ to the node $C_1$ and that
contains $u$ and $v$. Consider that there is at least one parent $p$ of $u$ that is not contained in
$C_2$. Thus there is a separator $K$ connecting $C_2$ toward $C_1$ in $T$ such that $K$ $p$-separates $p$
from all vertices in $C_2 \setminus K$. Note that on the chain from $C_1$ to $C_2$ in $T$, all separators must
contain $u$, otherwise they cannot separate $C_1$ from $C_2$. So, we have $u \in K$ but $v \notin K$ (if
$v \in K$, then $C_2$ is not the closest node of $T$ to the node $C_1$). In fact, for every parent $p'$ of $u$
that is contained in $C_1$ but not in $C_2$, $K$ separates $p'$ from all vertices in $C_2 \setminus K$, especially
the vertex $v$.

Define $S = \{\text{ant}(u \cup v) \cap (K \cup \{p \in pa(u) | p \in C_2\})\} \setminus \tau_u$, where $\tau_u$ is the chain component
that includes $u$. It is not difficult to see that $S$ is a subset of $C_2$. We need to show that $u$ and
v are p-separated by S, that is, every chain between u and v in G, say ρ, is blocked by S.

If ρ is not contained in ant(u ∪ v), then we obtain from Lemma 8 that ρ is blocked by S.

When ρ is contained in ant(u ∪ v), let x be adjacent to u on ρ, that is, ρ = (u, x, y, . . . , v).

We consider the three possible orientations of the edge between u and x. We now show that ρ is blocked in all three cases by S.

i: u ← x, so it is obvious that x is not a triplex node and we have two possible sub-cases:

1. x ∈ C2. In this case the chain ρ is blocked at x.

2. x ∉ C2. In this case K p-separates x from v. Theorem 4.8 guarantees that the set S′ = K ∩ ant(x ∪ v) also p-separates x from v. Note that S′ ∩ τu = ∅ to prevent a partially directed cycle, and S′ ⊆ S. So, S p-separates x from v i.e., the chain between v and x is blocked by S. Hence the chain ρ is blocked by S.

ii: u → x. We have the following sub-cases:

1. x ∈ ant(u). This case is impossible because a partially directed cycle would occur.

2. x ∈ an(v). This case is impossible because v cannot be a descendant of u.

iii: u — x, so x ∈ τu. In this case the chain ρ between u and v has a triplex node at y ∈ τu that is not in S. So, the chain ρ is blocked at y and cannot be activated by S.

□

Proof of Theorem 4.13. From (Cowell et al., 1999), we know that any separator S in junction tree T separates V1 \ S and V2 \ S in the triangulated graph ̃G′ V, where Vi denotes the variable set of the subtree Ti induced by removing the edge with a separator S attached, for i = 1, 2. Since the edge set of ̃G′ V contains that of undirected independence graph ̃G V for G, V1 \ S and V2 \ S are also separated in ̃G V. Since ̃G V is an undirected independence graph
for $G$, using the definition of $p$-separation tree we obtain that $T$ is a $p$-separation tree for $G$. □

Proof of Theorem 4.14. ($\Rightarrow$) If condition (i) is the case, nothing remains to prove. Otherwise, Lemma 10 implies condition (ii).

($\Leftarrow$) Assume that $u$ and $v$ are not contained together in any chain component and any node $C$ of $T$. Also, assume that $C_1$ and $C_2$ are two nodes of $T$ that contain $u$ and $v$, respectively. Consider that $C'_1$ is the most distant node from $C_1$, between $C_1$ and $C_2$, that contains $u$ and $C'_2$ is the most distant node from $C_2$, between $C_1$ and $C_2$, that contains $v$. Note that it is possible that $C'_1 = C_1$ or $C'_2 = C_2$. By the condition (i) we know that $C'_1 \neq C'_2$. The sufficiency of condition (i) is given by the definition of the $p$-separation tree, because any separator between $C'_1$ and $C'_2$ $p$-separates $u$ from $v$.

The sufficiency of conditions (ii) is trivial by the definition of $p$-separation. □

The following example shows that Theorem 4.14 cannot be strengthened.

Example 24. Consider the AMP CG $G$ in Figure E.1(a). Vertices $f$ and $h$ are not adjacent but both of them belong to the same chain component. As one can see in the Figure E.1(d), vertices $f$ and $h$ belong to nodes tree $C_1 = \{b, c, f, g, h\}$ and $C_2 = \{b, e, f, h\}$. However, none of them contains a subset of $V_G$ that $p$-separates $f$ from $h$.

Correctness of Algorithm 19. By the definition of $p$-separation trees and Theorem 4.14, the initializations at local and global skeleton recovery phases guarantee that no edge is created between any two variables which are not in the same node of the $p$-separation tree. Also, deleting edges at local and global skeleton recovery phases guarantees that any other edge between two $p$-separated variables can be deleted in some local skeleton or in the removal procedure at the global skeleton recovery phase. Thus the global skeleton obtained after line 22 is correct. Note that, in an AMP CG, every missing edge corresponds to at least one independency in the corresponding independence model. Therefore, each augmented edge $(u, v)$ in the undirected independence graph must be deleted at some subgraph over a node.
Figure E.1: (a) AMP CG $G$, (b) augmented graph $G^a$, (c) triangulated graph $(G^a)^t$, and (d) $p$-separation tree $T$.

of the $p$-separation tree or at some point of the removal procedure of the global skeleton recovery. The proof of the correctness of orientation rules R1-R4 can be found in (Peña, 2012). □