University of Warsaw
Faculty of Mathematics, Informatics and Mechanics

Damian Skrzypiec
Student no.: 320335

Structure Learning Algorithms for Chain Graphs

Master's thesis
in MATHEMATICS

Supervisor:
John Noble, PhD.
Institute of Applied Mathematics and Mechanics

December 2017

Supervisor's statement

Hereby I confirm that the present thesis was prepared under my supervision and that it fulfils the requirements for the degree of Master of Mathematics.

Date   Supervisor's signature

Author's statement

Hereby I declare that the present thesis was prepared by me and none of its contents was obtained by means that are against the law. The thesis has never before been a subject of any procedure of obtaining an academic degree. Moreover, I declare that the present version of the thesis is identical to the attached electronic version.

Date   Author's signature

Abstract

In this paper we present an algorithm for structural learning of chain graphs. Chain graphs are class of probabilistic graphical models which can be described as generalization of Bayesian networks and Markov random fields. We discuss in details LCD algorithm which is an algorithm for structural learning of chain graphs based on idea of decomposition chain graphs via separation trees. We also provide implementation of LCD algorithm available via R package *cglearn*.

Keywords

graphical model, chain graph, structure learning

Thesis domain (Socrates-Erasmus subject area codes)

11.2 Statistics

Subject classification

62 Statistics
62-09 Graphical methods
62E10 Characterization and structure theory

Title of the thesis in Polish

Algorytmy uczenia strukturalnego dla grafów łańcuchowych

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

The purpose of this project is to present algorithms for learning conditional independence structure of joint probability distributions represented by chain graphs. This is a special case of learning probabilistic graphical models which provides convenient representation of factorisation probability distribution using graphs. Two most common classes of probabilistic graphical models (PGMs) are Bayesian networks and Markov random fields. Bayesian networks are PGMs presented by directed acyclic graphs where Markov random fields are described by undirected graphs.

Chain graphs is a class of graphs that does not contains cycles (formal definition in 2.1.11). Chain graphs as class of probabilistic graphical models was introduced in 1984 by Lauritzen and Wermuth. It contains both directed and undirected edges in graph representation hence it is natural generalization of Bayesian networks and Markov random fields. Such a generalization was needed because of limitation of those two classes of PGMs. An edge in a Markov field model represent that there is a correlation between two random variables but it does not specify what type of correlation it is. On the other hand Bayesian network models contains only directed edges which represents only cause and effect relationships without possibility of existence of mutual correlation between two random variables.

The result of any probabilistic graphical model is a graph $G = (V, E)$ where vertices in set V corresponds to random variables and edges in set E presents some kind of conditional correlation between random variables. This is very convenient tool for data analysis in real-world applications. Nowadays companies accumulate huge amount of data. Very often major part of this data is noise or it is useless in further phase of modelling. Graphical models helps us discover idependence structure in our data set and therefore select appropriate subset. One of the greatest advantages of PGMs is easy of intepretability. Main disadvantage of most of PGMs is computational complexity. It is either impossible or it takes very long time to build PGM for data set with significant number of variables. Main algorithm described in this project focus on lower computational complexity of building chain graph through building smaller chain graphs and joining them in the right way.

Due to fact that Bayesian networks model only asymmetric casual relationships between variables and Markov random fields model only symmetric casual relationships beetwen variables one can face the problem when using one of those models would be inappropriate. The figure 1.1 presents Bayesian network which describes relationships beteen variables such as *Traffic Jam, Car Accident, Sirens, Bad Weather* and *Rush Hours*. In particular this model presents that traffic jams can be caused by some combination of factors like bad weather, car accidents and rush hours. It is possible that also traffic jams could cause car accidents. During traffic jams some of drivers could be frustrated and their behaviour could casue car

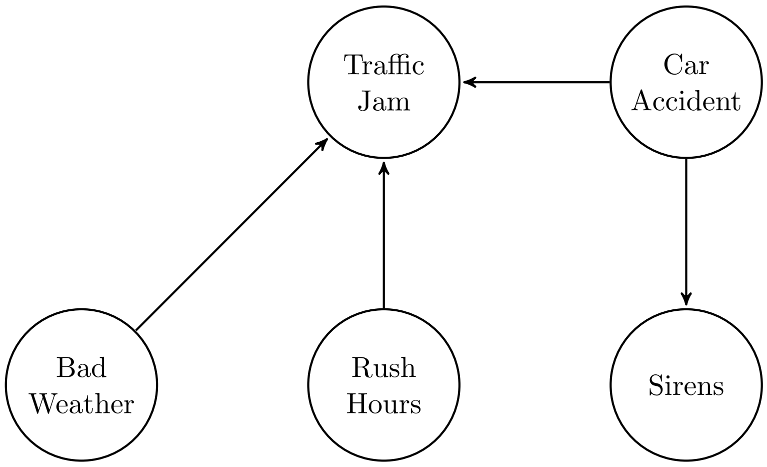

Figure 1.1: Example of Bayesian network

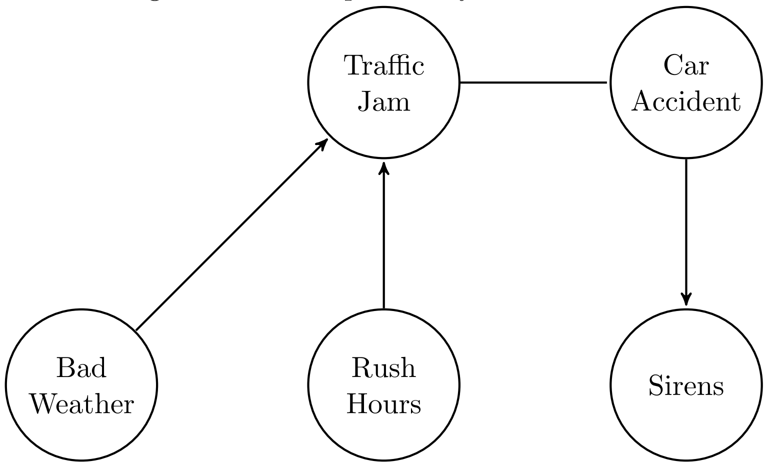

Figure 1.2: Example of chain graph

accidents. This type of correlation cannot be grasped by Bayesian networks but it can be done by using chain graph model. The figure 1.2 presents possible chain graph representation of the same data which was used for Bayesian network presented in the figure 1.1.

Rest of the paper is organized as follows. In chapter 2 we provide basic definitions from graph theory and graphical models. In the first subsection of chapter 3 we introduce concept of separation trees. In the second subsection of chapter 3 we describe LCD algorithm. We start from presenting mathematical basis for the algorithm. Next we illustrate both of phases of the algorithm. At the end of chapter 3 we estimate computational complexity of LCD algorithm. In chapter 4 we shortly specify the implementation of LCD algorithm and present functionality of *cglearn* package. In chapter 5 we present results of using LCD algorithm on some dataset.

Chapter 2

Preliminaries

2.1. Graph Theory Terminology

This section provides definitions of graph theory objects required for completeness of further sections. In this section, when is not mention different, V is default notation for set of graph's vertices and E is default notation for set of graph's edges.

Definition 2.1.1. (Undirected edge)
For vertices $u, v \in V$ we say that there is an undirected edge between vertices u and v if $(u, v) \in E$ and $(v, u) \in E$. Undirected edge between u and v is marked as $u - v$.

Definition 2.1.2. (Directed edge)
For vertices $u, v \in V$ we say that there is a directed edge from vertex u to vertex v if $(u, v) \in E$ and $(v, u) \notin E$. Directed edge from u to v is marked as $u \to v$.

Definition 2.1.3. (Parents, Neighbours, Boundry)
Let $G = (V, E)$ be a graph and $Y \subset V$ be a set of vertices. We define as follows

1. Parents of set Y in graph G is the set defined as $\mathrm{Pa}_G(Y) = \{X \ : \ X \to Z_Y \text{ for } Z_Y \in Y\}$.
2. Neighbors of set Y in graph G is the set defined as $\mathrm{Na}_G(Y) = \{X \ : \ X - Z_Y \text{ for } Z_Y \in Y\}$.
3. Boundry of vertex $v \in V$ in graph G is the set defined as $\mathrm{Bd}_G(v) = \mathrm{Pa}_G(v) \cup \mathrm{Ne}_G(v)$.

Definition 2.1.4. (Skeleton)
Skeleton of graph $G = (V, E)$ is a graph $G' = (V', E')$ where $V = V'$ and the set of edges E' is obtained by replacing directed edges of set E by undirected edges.

Definition 2.1.5. (Undirected complete graph)
Let V be a set of vertices. Graph $G = (V, E)$ is called undirected complete graph if set of edges E contains undirected edge between any two vertices from V.

Definition 2.1.6. (Route)
A *route* in graph $G = (V, E)$ is a sequence of vertices $(v_0, \ldots, v_k)$, $k \geq 0$, such that

$$(v_{i-1}, v_i) \in E \ \text{ or } \ (v_i, v_{i-1}) \in E$$

for $i = 1, \ldots, k$. The vertices v_0 and v_k are called *terminals.* A route is called descending if $(v_{i-1}, v_i) \in E$ for $i = 1, \ldots, k$. Descending route from u to v is marked as $u \mapsto v$. A route is called non descending if $(v_i, v_{i-1}) \in E$ or $(v_i, v_{i-1}) \in E \ \wedge \ (v_{i-1}, v_i) \in E$ for $i = 1, \ldots, k$. The subset of all vertices $v \in V$ which can be connected by a non descending route with vertex u is denoted by $\mathrm{An}(u)$.

Definition 2.1.7. (Path)
A route $r = (v_0, v_1, \ldots, v_k)$ in graph $G = (V, E)$ is called a path if all vertices in r are distinct.

Definition 2.1.8. (Complex)
A path $\pi = (v_1, v_2, \ldots, v_k)$ in graph $G = (V, E)$ is called complex if

1. $v_1 \rightarrow v_2$
2. $\forall_{i \in \{2,3,\ldots k-2\}}\ v_i - v_{i+1}$
3. $v_{k-1} \leftarrow v_k$
4. There is not additional edges in graph G for vertices in path π.

Vertices v_1 and v_k are called *parents* of the complex, set of vertices $\{v_2, v_3, \ldots, v_{k-1}\}$ is called *region* of the complex and number $k-2$ is the *degree* of the complex.

Next we define extended version of moral graphs. In Bayesian Networks moral graph is an undirected graph obtained from the original graph by adding undirected edges for not connected parents of the same child and then transform all edges into undirected edges. In case of chain graphs there can be situation when there are not connected parents of connected children (see 2.1). This situation can be interpreted as immoral and to be moralized a connection between parents are required.

Example 2.1.1. (Immorality in chain graph)

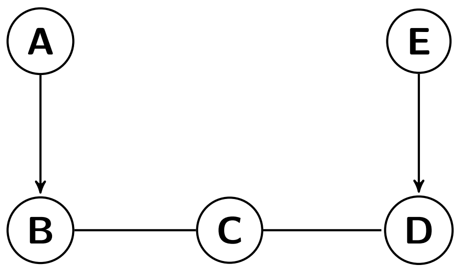

Figure 2.1: Immorality in chain graph

Definition 2.1.9. (Moral Graph)
Let $G = (V, E)$ be a graph. A moral graph $G^m = (V, E^m)$ of graph G is a graph obtained by firstly join parents of complexes in graph G and then replace all edges by undirected edges.

Definition 2.1.10. (Cycle)
A route $r = (v_0, v_1, \ldots, v_k)$ in graph $G = (V, E)$ is called a pseudocycle if $v_0 = v_k$ and a cycles if further route is a path and $k \geq 3$.

A graph with only directed edges is called an *undirected graph.* A graph without directed cycles and with only directed edges is called a *directed acyclic graph* (DAG).

Definition 2.1.11. (Chain graph)
A graph $G = (V, E)$ is called a chain graph if it does not have directed (pseudo) cycles.

Definition 2.1.12. (Section)
A subroute $\sigma = (v_i, \ldots, v_j)$ of route $\rho = (v_0, \ldots, v_k)$ in graph G is called section if σ is the maximal undirected subroute of route ρ. That means $v_i - \cdots - v_j$ for $0 \leq i \leq j \leq k$. Vertices v_i and v_j are called terminals of section σ. Further vertex v_i is called a head-terminal if $i > 0$ and $v_{i-1} \to v_i$ in graph G. Analogically vertex v_j is called a head-terminal if $j < k$ and $v_j \leftarrow v_{j+1}$ in graph G.

A section with two head-terminals is called *collider* section. Otherwise the section is called *non collider*. For a given set of vertices $S \subset V$ in graph G and section $\sigma = (v_i, \ldots, v_j)$ we say that section is hit by S if $\{v_i, \ldots, v_j\} \cap S \neq \emptyset$. Otherwise we say that section σ is outside set S.

Definition 2.1.13. (Intervention)
A route ρ in graph $G = (V, E)$ is blocked by a subset $S \subset V$ of vertices if and only if there exists a section σ of route ρ such that one of the following conditions is satisfied.

1. Section σ is a collider section with respect to ρ and σ is outside of S.

2. Section σ is non collider section with respect to ρ and σ is hit by S.

Lemma 2.1.1. Let $G = (V, E)$ be a chain graph, ρ be a route from vertex u to vertex v in G and W denote set of vertices of route ρ. If route ρ is blocked by $S \subset V$ in G such that $W \subset S$ then route ρ is blocked by W and any vertex set contaning W.

Proof. If route ρ is blocked by S then there is a non collider section of ρ which is hit by S or there is a collider section of ρ which is outside of S. Any collider section of ρ cannot be outside of S because $W \subset S$. With assumption that $W \subset S$ we know that every non collider section is hit by S and therefore is hit also by W and any other vertex set contaning set W.
$\blacksquare$

Example 2.1.2. (Graph definitions)
Based on the following two graphs (figures 2.2 and 2.3) we present examples of above defined definitions. Let graph presented in figure 2.2 be denoted as G. In graph G as example of descending route is (A, B, C, D) and example of non-descending route is (D, E, F, G). Graph G contains two complexes. Complex (A, B, C, D, E) is of degree equal to 3 and the other one (F, G, H, I) is of degree equal to 2. Graph G contains one cycle (I, J, K, I). The Route (F, G, H, I) in graph G contains section (G, H) which is head-to-head section.

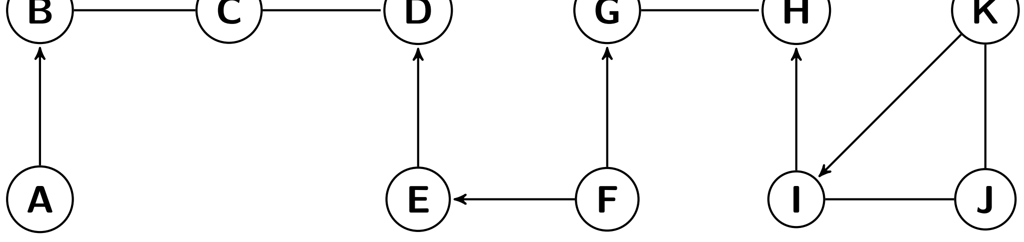

Figure 2.2: Example graph

Graph presented in figure 2.3 is moral graph of graph G. Additional undirected edges $A - E$ and $F - I$ are the result of connecting parents of complexes in the original graph G.

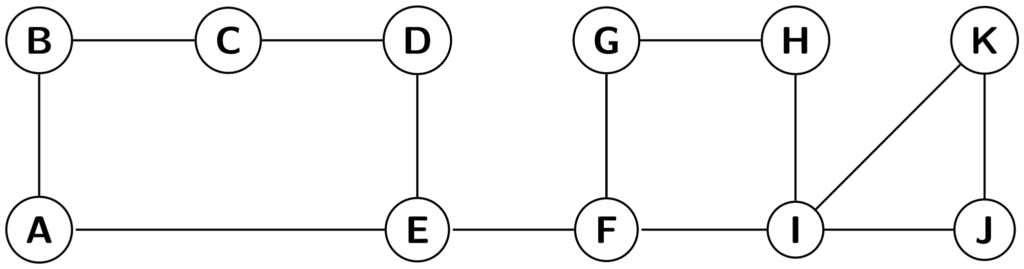

Figure 2.3: Moral graph of graph in figure 2.2

2.2. Graphical Model Terminology

Our main goal is to find an conditional independence structure of given joint probability distribution, hence we start from recalling definition of conditional independence.

Definition 2.2.1. (Conditional Independence)
Let $(X_1, X_2, \dots, X_n)$ be a random vector over probability space $(\Omega, \mathcal{F}, \mathbb{P})$. We say that random vectors $X_A = \{X_a \mid a \in A\}$ and $X_B = \{X_b \mid b \in B\}$ are conditional independent given $X_S = \{X_s \mid s \in S\}$ when for all $A_1, A_2, A_3 \in \mathcal{F}$

$$\mathbb{P}(X_A \in A_1, X_B \in A_2 \mid X_S \in A_3) = \mathbb{P}(X_A \in A_1 \mid X_S \in A_3)\mathbb{P}(X_B \in A_2 \mid X_S \in A_3) \tag{2.1}$$

where $A, B, S \subset 1, 2, \dots, n$. Conditional independence of X_A and X_B given X_S is denoted as $X_A \perp\!\!\!\perp X_B \mid X_S$.

The following definition of c-separation is an analogical version of d-separation, used in Bayesian Networks, for chain graphs. This definition was introduced by Studeny and Bouckaert in [7]. The notation c-separation is short of "chain separation" and it is written in this form to present analogy to definition of d-separation.

Definition 2.2.2. (c-separation)
Let $G = (V, E)$ be a chain graph. Let A, B, S be three disjoint subsets of the vertex set V, such that A and B are nonempty. We say that A and B are c-separated by S on G if every route within one of its terminals in A and the other in B is blocked by S. We call S a c-separator for A and B and mark as $\langle A, B \mid S\rangle_{\mathcal{G}}^{sep}$.

Definition 2.2.3. (faithfulness)
Let $G = (V, E)$ be a chain graph with random variables X_v associated with vertex $v \in V$. Let note domain of random variable X_v as $\mathcal{X}_v$. A probability measure $\mathbb{P}$ defined on $\prod_{v \in V} \mathcal{X}_v$ is *faithful* with respect to G if for any triple (A, B, S) of disjoint subsets of V where A and B are non-empty we have

$$\langle A, B \mid S\rangle_{\mathcal{G}}^{sep} \iff X_A \perp\!\!\!\perp X_B \mid X_S \tag{2.2}$$

In the same setup a probability measure $\mathbb{P}$ is called *Markovian* with respect to G if

$$\langle A, B \mid S\rangle_{\mathcal{G}}^{sep} \Longrightarrow X_A \perp\!\!\!\perp X_B \mid X_S \tag{2.3}$$

Definition 2.2.4. (independence model, I-map)
The independence model induced by a chain graph G, denoted as $I(G)$, is the set of separation statements $X \perp\!\!\!\perp Y \mid Z$ that holds in G. A chain graph H is an I-map (from independence map) of an independence model M if $I(H) \subset M$. Futhermore we say that H is an MI-map (minimal independence map) of M if after removing any edge from H it is not I-map of M anymore.

Remark 2.2.1. In the further section of this paper we will use c-separation statement for some chain graph G even if at that time graph G is unknown. In such a situation it should be interpreted as c-separation statement under probability distribution $\mathbb{P}$ which is faithful to graph G. In particular in the chapter 3 we will be using separation trees, which depends on chain graph G, to build chain graph G via LCD algorithm. Underlying meaning is that separation trees are built based on probability distribution that is faithful to chain graph G but we do not know form of graph G beforehand.

It is said that two chain graphs G and H are in the same Markov equivalence class if $I(G) = I(H)$. The following theorem from Frydenberg's paper [1] provides convenient tool for testing if two given chain graphs are the same in respect to Markov equivalence class.

Proposition 2.2.1. (Markov equivalence of chain graphs) [Theorem 5.6 from [1]]
Two chain graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ have the same Markov properties if and only if they same the same skeleton and the same complexes.

Chapter 3

LCD Algorithm

LCD algorithm is short for *Learning of Chain graphs via Decomposition* algorithm. The algorithm was introduced by Ma, Xie and Geng in paper *Structural Learning of Chain Graphs via Decomposition* [2]. LCD algorithm returns representative chain graph of it's Markov equivalence class, because even with perfect knowledge of data probability distribution any two chain graph structures within the same Markov equivalence class are indistinguishable. The algorithm is composed of two steps - recovering skeleton of chain graph and the second is recovering complexes. This idea is backed up by proposition 2.2.1. The main idea of skeleton recovery is to decompose set of variables into smaller sets, determine independence structure there and join results in a correct way. To decompose problem into smaller subproblems concept of separation trees is being used. The LCD algorithm required assumption about faithfulness of a probability distribution to some chain graph. This assumption is not very restrictive, because Jose Pena proved that the set of strictly positive probability distributions that are not faithful to some chain graph G has zero Lebesgue measure. Pena proved it for dicrete case in 2009 in [4] (theorem 5) and later in 2011 for gaussian case in [5] (theorem 4.2).

3.1. Decomposition of chain graphs

Concept of separation trees introduced in this section is the main tool in LCD algorithm for decompose original graph into smaller subgraphs. We will see in subsection 3.2.4 that computational complexity of LCD algorithm depends on size of largest node of input separation tree. In subsection 3.1.1 we introduce definition and example of node tree and separation tree. In subsection 3.1.2 we present method of construction separation trees.

3.1.1. Separation Trees

For graph $G = (V, E)$ we call set $\mathcal{C} = \{C_1, \dots, C_k\}$ as node set of graph G if $\mathcal{C}$ is a collection of distinct vertex sets such that $\forall i \in \{1, 2, \dots, k\}\, C_i \subset V$.

Definition 3.1.1. (Node Tree)
Let $G = (V, E)$ be a graph and $\mathcal{C} = \{C_1, \dots, C_k\}$ be a node set of graph G. A node tree is a graph $\mathcal{T}(G, \mathcal{C}) = (\mathcal{C} \cup \mathcal{S}, E)$, where $\mathcal{S} = \{C_i \cap C_j \mid i, j \in \{1, 2, \dots, k\}\}$ is set of so-called separators and $E = \{C_i - C_j \mid C_i \cap C_j \neq \emptyset \text{ and } i, j \in \{1, 2, \dots, k\}\}$ is set of undirected edges.

We will be using graphical convention for representing node trees proposed by Ma, Xie and Geng in [2]. Regular nodes (from set $\mathcal{C}$) in node tree will be displayed as triangles and separators (from set $\mathcal{S}$) will be displayed as rectangles.

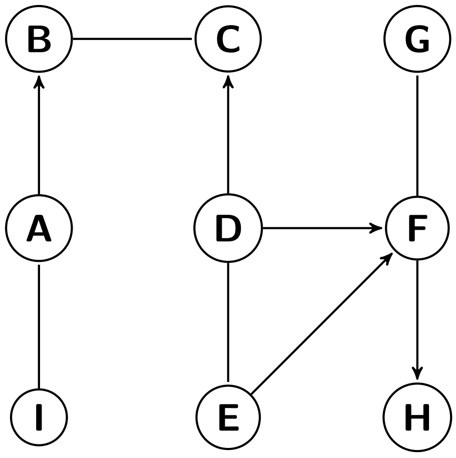

Figure 3.1: Example graph for node tree illustration

Example 3.1.1. (Node tree)
To illustrate node tree definition let's consider graph presented in figure 3.1 and node set $\mathcal{C} = \{\{A, I\}, \{A, B, C, D\}, \{D, E, F\}, \{D, F, G\}, \{E, F, H\}\}$.

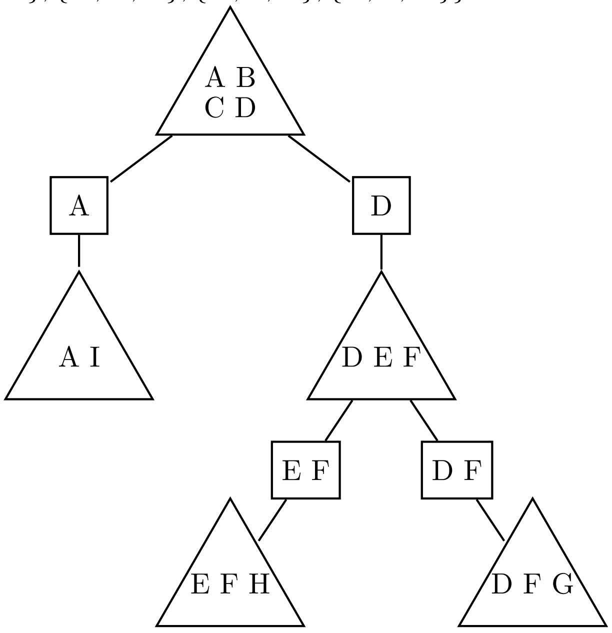

Figure 3.2: Node tree based on graph 3.1 and $\mathcal{C}$

Notice that if we remove separator S from node tree $\mathcal{T}(G, \mathcal{C})$ (or equivalently remove edge that contain separator S) then we got two separate node trees $\mathcal{T}(G, \mathcal{C}_1(S))$ and $\mathcal{T}(G, \mathcal{C}_2(S))$ where $\mathcal{C}_1(S) \cup \mathcal{C}_2(S) = \mathcal{C}$. To simplify notation we use

$$V_i(S) = \bigcup_{C \in \mathcal{C}_i(S)} C$$

as union of nodes from node tree $\mathcal{T}(G, \mathcal{C}_i(S))$ where $i \in \{1, 2\}$.

Definition 3.1.2. (Separation tree)
For given chain graph $G = (V, E)$ and node set $\mathcal{C}$ we say that node tree $\mathcal{T}(G, \mathcal{C})$ is a separation tree if

1. $\bigcup_{C\in\mathcal{C}} C = V$ and
2. for any separator S in node tree $\mathcal{T}(G,\mathcal{C})$ we have

$$\langle V_1(S)\setminus S, V_2(S)\setminus S \mid S\rangle_{\mathcal{G}}^{sep}$$

Let the node tree displayed in figure 3.2 be marked as $\mathcal{T}(G,\mathcal{C})$. The node tree $\mathcal{T}(G,\mathcal{C})$ contains four separators $\{A\}$, $\{D\}$, $\{E,F\}$, and $\{D,F\}$. At this point we know that $\bigcup_{C\in\mathcal{C}} C = V$. To examine if node tree $\mathcal{T}(G,\mathcal{C})$ is a separation tree we have to check if second condition from definition 3.1.2 are satisfied for every separator in $\mathcal{T}(G,\mathcal{C})$. If we consider separator $\{A\}$ of node tree $\mathcal{T}(G,\mathcal{C})$ then we obtain $V_1(\{A\}) = \{A,I\}$ and $V_2(\{A\}) = \{A,B,C,D,E,F,G,H\}$. The following condition of c-separation

$$\langle \{I\}, \{B,C,D,E,F,G,H\} \mid \{A\}\rangle_{\mathcal{G}}^{sep} \tag{3.1}$$

holds, because every route from $V_1(\{A\})\setminus\{A\}$ to $V_2(\{A\})\setminus\{A\}$ has only non head-to-head sections and for each route exist some section which is hit by A in graph G. Now if we consider separator $\{D\}$ we have $V_1(\{D\}) = \{A,B,C,D,I\}$ and $V_2(\{D\}) = \{D,E,F,G,H\}$. There is no route from $\{A,B,C,I\}$ to $\{E,F,G,H\}$ which would have a head-to-head section. Additionally every route from $\{A,B,C,I\}$ to $\{E,F,G,H\}$ contains section which is hit by D. Therefore $\{D\}$ c-separates $\{A,B,C,I\}$ and $\{E,F,G,H\}$ in graph G. Using the same argument we could prove that separators $\{E,F\}$ and $\{D,F\}$ also satisfies second condition in c-separation definition 3.1.2. Thus node tree $\mathcal{T}(G,\mathcal{C})$ presented in figure 3.2 is actual a separation tree of chain graph G presented in figure 3.1.

3.1.2. Construction of Separation Trees

In this subsection we present method for construction a separation trees by using labeled block ordering. Concept of labeled block ordering was introduced by Roverto and La Rocca in [6]. As the name indicates labeled block ordering is a set of blocks of vertices of chain graph where the ordering of blocks matters.

Definition 3.1.3. (labelled block ordering)
Let $G = (V,E)$ be a chain graph and $(V_i)_{i=1}^n$ be a partition of a set V. A labelled block ordering $\mathcal{B}$ of chain graph G is a sequence $(V_i^{l_i})_{i=1}^n$ such that $l_i \in \{d,g,u\}$ with convention $V_i = V_i^g$.

Definition 3.1.4. ($\mathcal{B}$-consistence)
Let $G = (V,E)$ be a chain graph and $\mathcal{B} = (V_i^{l_i})_{i=1}^n$ be a labelled block ordering of G. We say that G is $\mathcal{B}$-consistent if

1. every edge connecting vertices $A \in V_i$ and $B \in V_j$ is oriented from $A \to B$ for $i < j$;
2. for every i such that $l_i = u$, the subgraph G_{V_i} is a undirected graph;
3. for every i such that $l_i = d$, the subgraph G_{V_i} is a DAG;
4. for every i such that $l_i = g$, the subgraph G_{V_i} may have both directed and undirected edges.

Example 3.1.2. Let $V_1 = \{A,B,C,D,R,I\}$, $V_2 = \{G,F\}$ and $V_3 = \{H\}$. Figure 3.3 presents a $\mathcal{B}$-consistent labeled block ordering $\mathcal{B} = \{V_1^g, V_2^u, V_3^g\}$ of chain graph from figure 3.1.

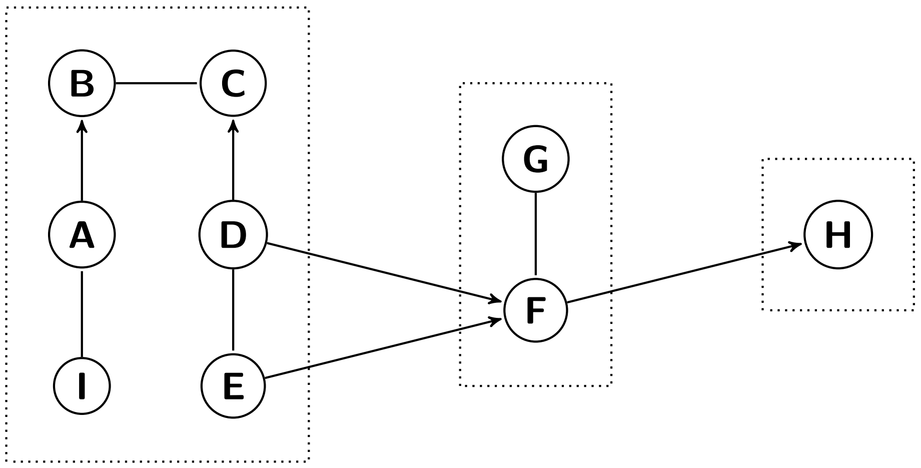

Figure 3.3: $\mathcal{B}$-consistent labeled block ordering for chain graph 3.1

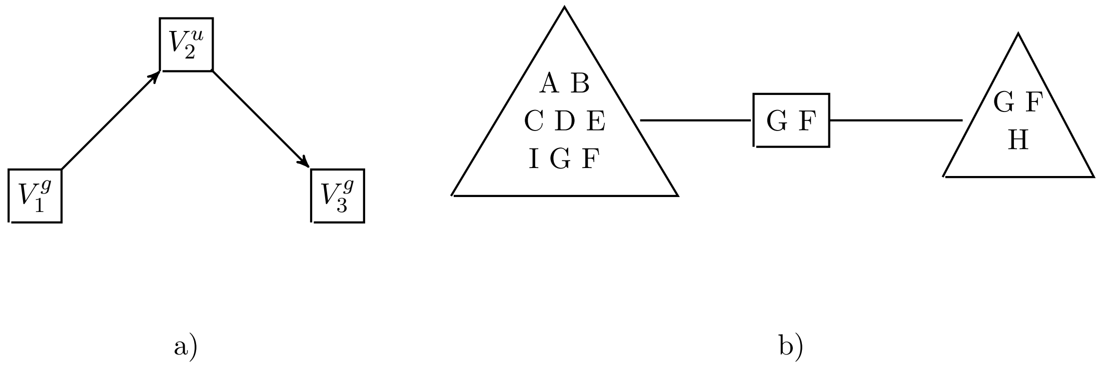

Figure 3.4: Result of Algorithm 1 for LBO from 3.3

Ma, Xie and Geng proposed in [2] algorithm for creating separation tree based on given labeled block ordering. Using the main feature of labeled block ordering which is known order of blocks one can construct a DAG of blocks and then create separation trees from DAG representation of blocks. Figure 3.4 presents execution of Algorithm 1 on labeled block ordering from previous example. On the lefthand side *a)* of the figure DAG of blocks is presented. It is result of first line of the algorithm. On the righthand side *b)* of the figure separation tree is presented which is built after second and third line of the algorithm.

Algorithm 1 (LCD) Separation Tree Construction with Labeled Block Ordering

```
Input: $\mathcal{B}$-consistent labeled block ordering $\mathcal{B} = (V_i^{l_i})_{i=1}^k$ of chain graph $G$
Output: Separation tree $\mathcal{T}$ of $G$.
1: Construct a DAG $D$ with blocks $(V_i)_{i=1}^k$
2: Construct a junction tree $\mathcal{T}$ by triangulating $D$
3: Replace each block $V_i$ in $\mathcal{T}$ by original vertices it contains
4: return: $\mathcal{T}$;
```

3.1.3. Separation Tree Lemmas

In this section we include lemmas which will be helpful in proving the main theorem for LCD algorithm. Most lemmas in this section was introduced in original paper about LCD algorithm [2]. The following lemma says that every route between two vertices that are separated by some separator in the separation tree is blocked by this separator. This lemma will be used in the proof of theorem 3.2.1.

Lemma 3.1.1. Let $G = (V, E)$ be a chain graph, $\mathcal{T}(G, \mathcal{C})$ be a separation tree and S be a separator of $\mathcal{T}(G, \mathcal{C})$. Suppose that $u \in V_1(S) \setminus S$, $u \in V_2(S) \setminus S$ and ρ is a route from u to v in G. Let W denote set of vertices of ρ. Then ρ is blocked by $W \cap S$ and any vertex set contaning $W \cap S$.

Proof. We can assume that route ρ is of the form

$$\rho = \left(\overbrace{v_1^1, v_2^1, \ldots, v_k^1}^{V_1(S)\setminus S}, \underbrace{s_1, \ldots, s_n}_{S}, \overbrace{v_1^2, v_2^2, \ldots v_m^2}^{V_2(S)\setminus S} \right) \tag{3.2}$$

Otherwise every vertex of ρ would be in either $V_1(S) \setminus S$ or in $V_2(S) \setminus S$. That would imply existance of two adjacent vertices $v \in V_1(S) \setminus S$ and $w \in V_2(S) \setminus S$ and further that would contradictory to condition from separation tree definition $\langle V_1(S) \setminus S, V_2(S) \setminus S \mid S \rangle_{\mathcal{G}}^{sep}$.

Let ψ be the subroute of the vertices $(v_k^1, s_1, \ldots, s_n, v_1^2)$ and W_ψ be the vertex set of ψ with vertices $\{v_k^1, v_1^2\}$ excluded. Since $W_\psi \subset S$ there is at least one non collider section of route ψ and, from lemma 2.1.1, every non collider section on ψ is hit by W_ψ. Therefore route ρ is blocked by set S or any set contaning S when ψ is part of non collider section of ρ. To complete proof we have to show that thesis is satisfied in case when subroute ψ is part of collider section of route ρ. From now on we can assume that route ρ has collider section σ of the form

$$\sigma = v_i^1 \rightarrow v_{i+1}^1 - \cdots - v_k^1 - s_1 - \cdots - s_n - v_1^2 - \ldots v_j^2 \leftarrow v_{j+1}^2 \tag{3.3}$$

for some $i \in \{1, 2, \ldots k\}$ and $j \in \{1, 2, \ldots, m\}$. We have to consider few corner cases. In case when $v_i^1 \in S$ then exists a non collider section of ρ which is blocked by S or any set contaning S. In case when $v_i^1 \in V_1(S) \setminus S$ and $v_{j+1}^2 \in S$ then exists a non collider section of ρ which is blocked by S or any set contaning S. In case when $v_{j+1}^2 \in V_1(S) \setminus S$ then we can consider a non collider section of route ρ which starts from v_{j+1}^2. Now it is sufficient to consider regular case when $v_i^1 \in V_1(S) \setminus S$ and $v_{j+1}^2 \in V_2(S) \setminus S$. This case cannot occure in separation tree because that would imply $v_i^1 \not\perp\!\!\!\perp v_{j+1}^2 \mid S$ which is contradictory to condition from separation tree definition $\langle V_1(S) \setminus S, V_2(S) \setminus S \mid S \rangle_{\mathcal{G}}^{sep}$. ■

Lemma 3.1.2. Let $G = (V, E)$ be a chain graph, $\mathcal{T}(G, \mathcal{C})$ be a separation tree of chain graph G and $u, v \in V$ are non adjacent vertices in G. If ρ is a route from u to v is not contained in $\mathrm{An}(u) \cup \mathrm{An}(v)$, then ρ is blocked by any subset of $\mathrm{An}(u) \cup \mathrm{An}(v)$.

Proof. Since route ρ is not contained in set $\mathrm{An}(u) \cup \mathrm{An}(v)$ there exists vertices s_1, s_2, t_1, t_2 such that route ρ is of the form

$$\begin{cases} \rho = (u, \ldots, s_1, s_2, \ldots, t_1, t_2, \ldots, v) \\ \{s_1, t_2\} \in \mathrm{An}(u) \cup \mathrm{An}(v) \\ \{s_2, t_1\} \cap (\mathrm{An}(u) \cup \mathrm{An}(v)) = \emptyset \end{cases} \tag{3.4}$$

In this case we have $s_1 \rightarrow s_2$ and $t_1 \leftarrow t_2$, because otherwise s_2 or t_1 would be in set $\mathrm{An}(u) \cup \mathrm{An}(v)$. Therefore there exists at least one collider section between vertices s_1 and t_2 on route ρ that is outside of $\mathrm{An}(u) \cup \mathrm{An}(v)$. Thus route ρ is blocked by any subset of $\mathrm{An}(u) \cup \mathrm{An}(v)$. ■

Lemma 3.1.3. Let $G = (V, E)$ be a chain graph, $\mathcal{T}(G, \mathcal{C})$ be a separation tree of chain graph G and u, v are two adjacent vertices. Then there exists node $C \in \mathcal{T}(G, \mathcal{C})$ which contains both vertices u and v.

Proof. Let suppose opposite. In this case there exists separator K in $\mathcal{T}(G), \mathcal{C}$ such that $u \in V_1(K)$ and $v \in V_2(K)$. And by separation tree definition this implies that $\langle u, v \mid K\rangle_{\mathcal{G}}^{sep}$ and it is not possible becasue u and v are adjacent vertices. ■

The following lemma 3.1.4 states that there exists some set S which c-separates some non adjacent vertices u and v when both of vertices u and v are not contained in the separator connected to their node. This lemma will be used in the proof of theorem 3.2.1.

Lemma 3.1.4. Let $G = (V, E)$ be a chain graph and $\mathcal{T}(G, \mathcal{C})$ be a separation tree for $\mathcal{C} = \{C_1, \dots, C_k\}$. Let u and v be a non adjacent vertices in G such that $u, v \in C_i$ for some $i \in \{1, \dots, k\}$. If u and v are not contained in the same separator connected to C_i then there exists a subset S of C_i such that $\langle u, v \mid S\rangle_{\mathcal{G}}^{sep}$.

Proof. We show that the following set of vertices

$$S = (\mathrm{An}(u) \cup \mathrm{An}(v)) \cap (C_i \setminus \{u, v\}) \tag{3.5}$$

is the c-separator of vertices u and v. Let ρ be a route from u to v in graph G. If route ρ is not contained in set $\mathrm{An}(u) \cup \mathrm{An}(v)$ then ρ is blocked by S, by lemma 3.1.2. Therefore from now on we can assume that route ρ is contained in set $\mathrm{An}(u) \cup \mathrm{An}(v)$. To show that S is c-separator we consider the following cases of first section of route ρ containing vertex u.

1. $u - s_1 - \cdots - s_n \leftarrow x$ where $x \neq v$ and $x \in C_i$.
   In this case $x \in \mathrm{An}(u)$ and $x \in C_i$, therefore $x \in S$. That implies existance of non collider section containing x which is hit by S. Thus u and v are c-separated by S.

2. $u - s_1 - \cdots - s_n \rightarrow x - t_1 - \cdots - t_m \rightarrow y$ where $x \neq v$ and $x \in C_i$.
   In this case $x \in \mathrm{An}(u) \cup \mathrm{An}(v)$ but $x \notin \mathrm{An}(u)$, thus $x \in \mathrm{An}(v)$. Because of $x \in C_i$ and $x \in \mathrm{An}(v)$ we know that $x \in S$. Similar to the previous case vertices u and v are c-separated because there is at least one non collider section of route ρ containing x which is hit by S.

3. $u - s_1 - \cdots - s_n \rightarrow x - t_1 - \cdots - t_m \leftarrow y$ where $x \neq v$ and $x, y \in C_i$.
   In this case $x \notin \mathrm{An}(u)$ and still route ρ is contained in $\mathrm{An}(u) \cup \mathrm{An}(v)$ therefore $x \in \mathrm{An}(v)$ and thus $y \in \mathrm{An}(v)$. That implies $x, y \in S$. Similar to previous cases there is at least one non collider section containing x which is hit by S.

4. $u - s_1 - \dots s_n \rightarrow x - t_1 - \cdots - t_m \leftarrow y$ where $x \neq v$, $x \in C_i$ and $y \notin C_i$.
   In this case we assume that $y \in C_j$ where $j \neq i$. Next we assume that y belongs to the separator K connected C_i and next node on the path from C_i to C_j on separation tree $\mathcal{T}(G, \mathcal{C})$. To resolve this case we divide it into the following cases

   (a) In case when $\{u, s_1, \dots, s_n\} \cap S \neq \emptyset$ then non collider section $u - s_1 - \cdots - s_n$ is hit by S.

   (b) In case when $\{u, s_1, \dots, s_n\} \cap S = \emptyset$ and $\{x, t_1, \dots, t_k\} \cap S \neq \emptyset$ then collider section $x - t_1 - \cdots - t_k$ is outside S and therefore route ρ is blocked.

(c) In case when $\{u, s_1, \ldots, s_n\} \cap S = \emptyset$ and $\{x, t_1, \ldots, t_k\} \cap S \neq \emptyset$ then there is some $t_j \in \{t_1, \ldots, t_k\}$ such that $t_j \neq v$ and $t_j \in C_i \cap \mathrm{An}(v)$. Since $\{u, s_1, \ldots, s_n\} \cap S = \emptyset$ and $u \perp\!\!\!\perp y \mid K$ there should be no collider sections in route $(u, s_1, \ldots, s_n, x, t_1, \ldots, t_m)$, therefore this case can never happen.

Now we consider case when $v \notin K$. Because of condition $\{x, t_1, \ldots, t_m\} \subset \mathrm{An}v$ we know that there exists at least one arrow '$\rightarrow$' on sub-route of ρ from y to v. Futhermore there is no further arrow '$\leftarrow$' closer to ther right end v than it is. Therefore this case identical as one of cases 1., 5. or 6. with u replaced by v.

5. $u - s_1 - \cdots - s_m - v$, $u - s_1 \cdots - s_m \rightarrow v$ or $u - s_1 - \cdots - s_m \leftarrow v$.
We know that $s_1 \neq v$ and $s_m \neq u$ because vertices u and v are not adjacent. Suppose that $\{s_1, \ldots, s_m\} \cap S = \emptyset$. That implies, based on definition of set S, $\{s_1, \ldots, s_m\} \cap C \subset \{u, v\}$. In case when $\{s_1, \ldots, s_m\} \cap S \subset \{u\}$ then we have non collider section of route ρ which is hit by S and therefore it is blocked by S. We know that $s_1 \neq u$ and in case when $\{s_1, \ldots, s_m\} \cap C \subset \{v\}$ then we have $s_1 \notin C$. Suppose that $s_1 \in C_j$ where C_j is a node tree in separation tree $\mathcal{T}(G, \mathcal{C})$ which is different from node tree C_i and further suppose that K is a separator connected to node C_i on the path between C_i and C_j. In this set up $u \in K$, $v \notin K$ and $s_1 \perp\!\!\!\perp v \mid K$. However since $\{s_1, \ldots, s_m\} \cap C \subset \{v\}$ we have $\{s_1, \ldots, s_m\} \cap K = \emptyset$ which is not possible. Therefore we have $\{s_1, \ldots, s_m\} \cap S \neq \emptyset$ and route ρ is blocked.

6. $u - s_1 - \cdots - s_m \rightarrow x$ or $u - s_1 - \cdots - s_m \leftarrow x$ where $x \notin C_i$.
We consider the first case $u - s_1 - \cdots - s_m \rightarrow x$. If $\{u, s_1, \ldots, s_m\} \cap S \neq \emptyset$ then route ρ is blocked by S because there is a non collider section of ρ which is hit by S. In the opposite case ($\{u, s_1, \ldots, s_m\} \cap S = \emptyset$) suppose that $x \in C_j$ where C_j is a node tree in separation tree $\mathcal{T}(G, \mathcal{C})$ which is different from node tree C_i and further suppose that K is a separator connected to node C_i on the path between C_i and C_j. Now we consider the following two cases.

   (a) We assume that $v \in K$. This assumption implies $u \notin K$ and $\{u, s_1, \ldots, s_m\} \cap K \subset \{v\}$. If $\{u, s_1, \ldots, s_m\} \cap K = \{v\}$ then this case is reduced to the case 5.. If $\{u, s_1, \ldots, s_m\} \cap K = \emptyset$ then $u \not\perp\!\!\!\perp x \mid K$ and that is contradictory to the definition of separation tree.

   (b) We assume that $v \notin K$. Using similar deduction as in the previous case we have $u \in K$. Since any non collider section of sub-route of route ρ is also non collider section of route ρ we consider sub-route of route ρ starting in x. Let W denote set of vertices of the sub-route excluding the two end vertices. We know that $W \cap K \subset S \cup \{u, v\}$. In case when $W \cap K \subset S$, by lemma 3.1.1, route ρ is blocked by S. In case when non collider section of sub-route is hit by u or v we can consider modified sub-route starting at that point. Applying the same approach using lemma 3.1.1 to the new sub-route we have that ρ is blocked by S.

Exactly the same approach is used to the other condition in case 6. We have considered all the possible cases. Now proof is complete. ■

3.2. Algorithm

We pointed out before that LCD algorithm is composed of two phases. In the first phase skeleton of chain graph is constructed based on provided separation tree of chain graph and knowledge about conditional probability distribution. Outcome of this phase is skeleton $G^{'}$ of chain graph G and set of separators $\mathcal{S}$. In the second phase complexes of chain graph G are reconstructed. Algorithm in the second phase is based on outcome of the first phase. Mathematical grounds for both algorithms are described in subsection 3.2.1. In subsection 3.2.2 we describe skeleton recovery algorithm (phase 1) and present associated example. In subsection 3.2.3 we describe complex recovery algorithm (phase 2) and also present associated example. In subsection 3.2.4 we performe analysis of computitional complexity of the LCD algorithm.

3.2.1. Mathematical basis

In this subsection we introduce mathematical basis for correctness of LCD algorithm. Theorem 3.2.1 provide charaterization of c-separation condition in the separation trees. Based on the theorem 3.2.1 skeleton recovery phase of LCD algorithm was constructed.

Theorem 3.2.1. ([2], chapter 3.1, Theorem 3)
Let $\mathcal{T}(G, \mathcal{C})$ be a separation tree for chain graph G. Then vertices u and v are c-separated by some set $S_{uv} \subset V$ in G $(*)$ if and only if one the following conditions hold:

1. Vertices u and v are not contained together in any node C of $\mathcal{T}(G, \mathcal{C})$,
2. Vertices u and v are contained together in some node C, but for any separator S connected to C, $\{u, v\} \not\subset S$, and there exists $S^{'}_{uv} \subset C$ such that $\left\langle u, v \mid S^{'}_{uv} \right\rangle^{sep}_{\mathcal{G}}$,
3. Vertices u and v are contained together in some node C and both of them belong to some separator connected to C, but there is a subset $S^{'}_{uv}$ of either $\bigcup_{u \in C^{'}} C^{'}$ or $\bigcup_{v \in C^{'}} C^{'}$ such that $\left\langle u, v \mid S^{'}_{uv} \right\rangle^{sep}_{\mathcal{G}}$.

Proof. At first we will prove the sufficiency of conditions (1), (2) and (3).

(1) $\Rightarrow$ $(*)$.
In this case we assume that vertices $u \in C_i$ and $v \in C_j$ are in different nodes of separation tree $\mathcal{T}(G, \mathcal{C})$. From lemma 3.1.1 we know that every route between vertices u and v is blocked by any separator in separation tree $\mathcal{T}(G, \mathcal{C})$ between nodes C_i and C_j. Therefore there is at least one set which c-separates u and v in G.

Implications (2) $\Rightarrow$ $(*)$ and (3) $\Rightarrow$ $(*)$ are satisfied by definition of c-separation.

$(*) \Rightarrow (1) \vee (2) \vee (3)$.
Now we assume that there is some set $S \subset V$ which c-separates vertices u and v in graph G. We can also assume that vertices u and v are contained in the same node of separation tree. Otherwise using lemma 3.1.1 condition (1) is satisfied. When u and v are contained in the same node $C \in \mathcal{C}$ we have to consider two cases. The first case is when vertices u and v are not contained in the same separator connected to node C. The second case is otherwise. The first case is proven by lemma 3.1.4. It is know that in chain graph G for any two distinct vertices u and v we have

$$\langle u, v \mid \mathrm{bd}_G(u) \cup \mathrm{bd}_G(v) \rangle^{sep}_{\mathcal{G}} \tag{3.6}$$

Therefore in the second case we have $\mathrm{bd}_G(u) \subset \bigcup_{u \in C} C$ and $\mathrm{bd}_G(v) \subset \bigcup_{v \in C} C$. That implies at least one of conditions 2 and 3 holds. ■

The following theorem 3.2.2 helps to narrow down number of possibilities to verify in the complex recovery phase of the LCD algorithm.

Theorem 3.2.2. ([2], chapter 3.1, Proposition 4)
Let G be a chain graph and $\mathcal{T}(G, \mathcal{C})$ be a separation tree of G. For any complex $\mathcal{K}$ in G, there exists some node tree C of $\mathcal{T}(G, \mathcal{C})$ such that $\mathcal{K} \subset C$.

Proof. Let suppose opposite. Let $\mathcal{K} = (u, w_1, \ldots, w_n, v)$ be a complex in chain graph G such that for any node C in separation tree $\mathcal{T}(G, \mathcal{C})$ vertex u and v are not contained together in C. Now let suppose that $u \in C_u$ and $v \in C_v$ where C_u and C_v are nodes of the separation tree.

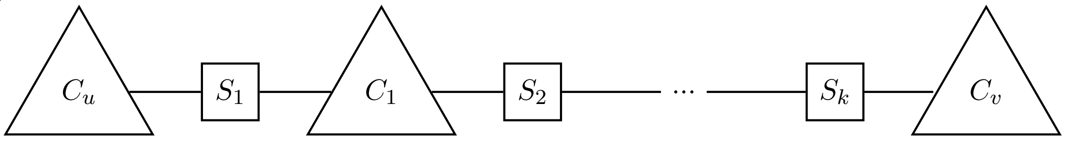

Figure 3.5: Path between C_u and C_v in separation tree $\mathcal{T}$

Futhermore we introduce $\rho = \{C_u, S_1, C_1, S_2, \ldots, S_k, C_v\}$ as a path between C_u and C_v in the separation tree $\mathcal{T}(G, \mathcal{C})$. We observe that $u \notin S_1$ and $v \notin S_1$ then we have $\{w_1, w_2, \ldots, w_n\} \cap S_1 \neq \emptyset$ and therefore $u \not\perp\!\!\!\perp v \mid S_1$. The last implication holds because in this case we have a head-to-head section $u \rightarrow w_1 - w_2 - \cdots - w_n \leftarrow v$ and if $\{w_1, w_2, \ldots, w_n\} \cap S_1 \neq \emptyset$ then this section is not outside of S_1. Condition $u \not\perp\!\!\!\perp v \mid S_1$ is contrary to definition of separation tree. Because of the above contrary we can assume that $u \in C_1$ and repeate the same process. After finite number of iteration we obtain that u and v have to be contained in the same node of the separation tree. ■

3.2.2. Skeleton recovery

Skeleton Recovery algorithm is composed of three parts. In the first part of this algorithm (lines 3-11) local skeletons are recovered. This is obtained by looping over all nodes in $\mathcal{T}(G, \mathcal{C})$. For given node $C_h \in \mathcal{T}(G, \mathcal{C})$ we create complite graph $G_h = (C_h, E_h)$ and we test conditional independence of all possible pairs of C_h. For fixed pair $\{u, v\} \in C_h$ we every subset $S_{uv} \subset C_h$ we test condition $u \perp\!\!\!\perp v \mid S_{uv}$. If such a condition is satisfied then edge (u, v) is removed from local graph G_h. By condition 1 from theorem 3.2.1 we observe that edge which is removed from local skeleton cannot occure in global skeleton. This observation increase performance of implementation of Skeleton Recovery algorithm. In the second part of Skeleton Recovery algorithm (lines 12-17) local skeletons are combined into global skeleton and some of extra edges are removed based on second condition in theorem 3.2.1. The third part of this algorithm (lines 18-24) also focus on removing incorrect edges. This part of algorithm are backed up by third condition of theorem 3.2.1. The following algorithm was introduced in [2], chapter 3.2, algorithm 1.

Example 3.2.1. To illustrate execution of the LCD algorithm let's assume we have data which joint distribution is faithful to graph presented in figure 3.1 but we do not know it's chain graph representation yet. Additionally we have separation tree presented on figure 3.2. Result of first phase of skeleton recovery algorithm is presented on figure 3.6. For every node tree we have local undirected graph representing local (in sense of particular node) independence structure. Phase two of skeleton recovery algorithm join local graphs into global undirected graph and removes some of redundant edges. Result of execution second phase is represented on figure 3.7. Result of applying skeleton recovery algorithm is the same

Algorithm 2 (LCD) Skeleton Recovery

Input: A separation tree $\mathcal{T}(G,\mathcal{C})$; perfect conditional independence knowledge about $\mathbb{P}$.
Output: The skeleton $G^{'}$ of G; a set $\mathcal{S}$ of c-separators.

```
 1: procedure RecoverySkeleton($\mathcal{T}(G,\mathcal{C})$)
 2:     $\mathcal{S} = \emptyset$
 3:     for all node $C_h \in \mathcal{T}(G,\mathcal{C})$ do
 4:         Create complete undirected graph $G_h = (C_h, E_h)$;
 5:         for all vertex pair $\{u,v\} \subset C_h$ do
 6:             if $\exists S_{uv} \subset C_h$ $u \perp\!\!\!\perp v \mid S_{uv}$ then
 7:                 Delete edge $(u,v)$ from graph $G_h$;
 8:                 $\mathcal{S} := \mathcal{S} \cup S_{uv}$;                    ▷ Add set $S_{uv}$ to separators
 9:             end if
10:         end for
11:     end for
12:     Combine all the graphs $(G_h)_{i\in\{1,\ldots,H}$ into undirected graph $G^{'} = (V, \bigcup_{h=1}^{H} E_h)$;
13:     for all $\{u,v\} \in G^{'}$ contained in more then one node of $\mathcal{T}(G,\mathcal{C})$ do
14:         if $\exists C_h$ $\{u,v\} \subset C_h$ and $(u,v) \notin E_h$ then
15:             Delete the edge $(u,v)$ from $G^{'}$;
16:         end if
17:     end for
18:     for all $\{u,v\} \in G^{'}$ contained in more then one node of $\mathcal{T}(G,\mathcal{C})$ do
19:         $N_{uv} := \{S \subset \mathrm{ne}_{G'}(u) \cup \mathrm{ne}_{G'}(v) \mid S \not\subset C_h$ and $\{u,v\} \subset C_h\}$
20:         if $u \perp\!\!\!\perp v \mid S_{uv}$ for some $S_{uv} \subset N_{uv}$ then
21:             Delete edge $(u,v)$ from graph $G^{'}$;
22:             $\mathcal{S} := \mathcal{S} \cup S_{uv}$;                        ▷ Add set $S_{uv}$ to separators
23:         end if
24:     end for
25:     return: $G^{'}$, $\mathcal{S}$.
26: end procedure
```

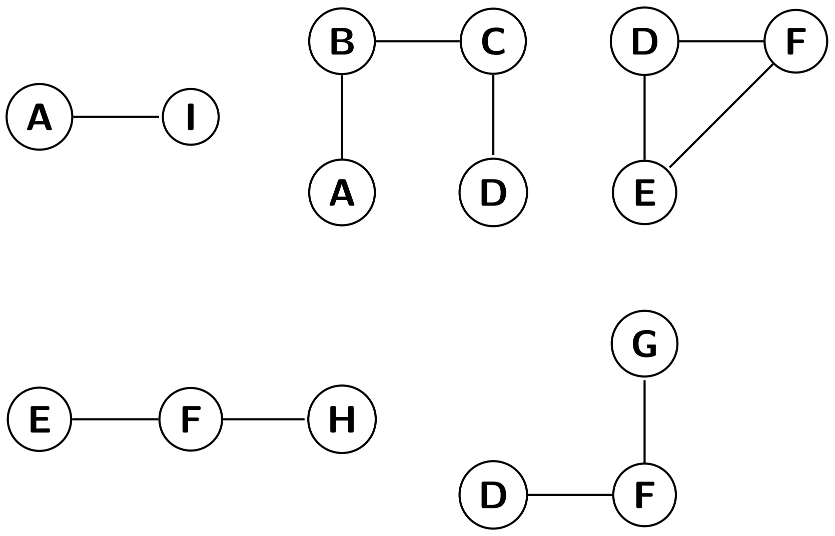

Figure 3.6: Result of first phase of LCD algorithm

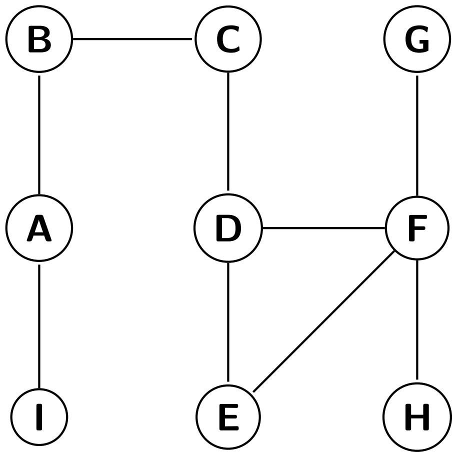

Figure 3.7: Result of second phase of LCD algorithm

as outcome from second phase of the algorithm, because there isn't pair of random variable satisfying condition from 20th line of Algorithm 2. Output set of separators in this example is $\mathcal{S} = \{\{B, C\}, \{F\}\}$.

3.2.3. Complex recovery

Complex Recovery algorithm is based on output of the Skeleton Recovery algorithm. This algorithm is far less computationaly complex than the first part of the LCD algorithm. Especially because set of separators from Sekelton Recovery algorithm narrows down the number of possible candidates. The following algorithm was introduced in [2], chapter 3.3, algorithm 2.

By the theorem 3.2.2 and lemma 3.1.3 we know that all candidates to edge orientation are considered in line 3 of the algorithm 3.

Example 3.2.2. In this example we present performance of complex recovery algorithm for outcomes from skeleton recovery algorithm presented in Example 3.2.1. If we consider pair $[D, A]$ in outer loop in the algorithm we find that $S_{DA} = \{B\}$ and $D \not\perp\!\!\!\perp A \mid \{B\} \cup \{C\}$,

Algorithm 3 (LCD) Complex Recovery

Input: Perfect conditional independence knowledge about $\mathbb{P}$; the skeleton $G^{'}$ and the set $\mathcal{S}$ of c-separators obtained in algorithm 2.
Output: The pattern G^{*} of graph G.

```
1: procedure ComplexRecovery($\mathcal{T}(G, \mathcal{C})$)
2:     Initialize $G^{*} = G^{'}$
3:     for all ordered pair $[u, v]$ : $S_{uv} \in \mathcal{S}$ do
4:         for all $u - w$ in $G^{*}$ do
5:             if $u \not\perp\!\!\!\perp v \mid S_{uv} \cup \{w\}$ then
6:                 Orient $u - w$ as $u \to w$ in $G^{*}$;
7:             end if
8:         end for
9:     end for
10:    return: Pattern of $G^{*}$.
11: end procedure
```

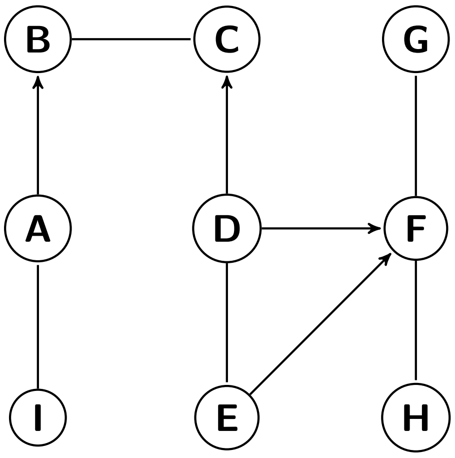

Figure 3.8: Result of Complex Recovery Algorithm

therefore we orient edge $D \to C$. Similar we orient edge $A \to B$, because for pair $[A, D]$ in the outer loop we have $S_{AD} = \{B\}$ and $A \not\perp\!\!\!\perp D \mid \{B\}$. Conditional independence in this case is not satisfied because condition $\langle A, D \mid B \rangle_{G}^{sep}$ does not hold and we have assumption of faithfulness. For $[D, G]$ in outer loop we have $S_{DG} = \{F\}$ and $D \not\perp\!\!\!\perp G \mid S_{DG} \cup \{F\}$, hence we orient edge $D \to F$. Orientation of edge $E \to F$ is obtained by consideration $[E, H]$ in outer loop and $w = F$. Result of complex recovery algorithm is presented in figure 3.8. The edge $[F, H]$ was not oriented by the algorithm because condition $F \not\perp\!\!\!\perp H \mid F$ is not satisfied. As we mentioned before LCD algorithm creates representative of Markov equivalence class of given chain graph. Chain graphs presented in figure 3.8 and 3.1 are in the same Markov equivalence class.

3.2.4. Algorithm complexity

The skeleton and complex recovery phases are independent in sense of computational complexity, hence we can find upper bounds for those two phases separately. Let suppose that we have unknown chain graph $G = (V, E)$ with $|V| = n$ and $|E| = m$. Let further suppose that the input separation tree $\mathcal{T}$ contains tree nodes $H = \{C_1, C_2, \ldots, C_k\}$. To denote number of elements of separation tree node C_i we use c_i and by m we denote count of the biggest node in a separation tree $\mathcal{T}$, that is $m = \max\{c_i \mid i \in \{1, 2, \ldots, k\}\}$.

The most computational expensive step in the skeleton recovery algorithm is loop in line 5 and verifying condition in line 6 of Algorithm 2. For given node in the separation tree C_i looping over all pairs $\{u, v\} \subset C_i$ is of complexity $\frac{1}{2}c_i \cdot (c_i + 1)$ which is $\mathcal{O}(c_i^2)$. For given node in the separation tree and given pair of vertex $\{u, v\}$ verifying condition in line 6 of Algorithm 2 is of cost 2^{c_i} because it requires to look over all subsets of C_i. Second and third steps of the skeleton recovery algorithm are less computational complex then the first step. Therefore complexity of the whole algorithm is determined by complexity of the first step which can be estimated as follow

$$\begin{aligned} T(\mathcal{T}) &= \mathcal{O}\left(\sum_{C_i \in H} \frac{1}{2} c_i (c_i + 1) 2^{c_i}\right) \leq \\ &\leq \mathcal{O}\left(\sum_{C_i \in H} \frac{1}{2} m (m + 1) 2^{m}\right) = \\ &= \mathcal{O}\left(k m^2 2^m\right) \end{aligned} \tag{3.7}$$

In the complex recovery phase we have double loop and verifying conditional independence condition. The first loop is over pairs of vertices $\{u, v\}$ which are connected, therefore number of iteration can be bounded by $\mathcal{O}(m)$. In the second loop for each pair $\{u, v\}$ there is at most n vertices to check. Thus computational cost for double loop in the complex recovery algorithm is $\mathcal{O}(nm)$. Verifying conditional independence condition in line 5 of algorithm 3 for set vertices u, v and w takes $\mathcal{O}(1)$. The computational complexity of this phase of LCD algorithm is far more less than the complexity of the skeleton recovery. Therefore computational cost of the LCD algorithm is dominated by computational cost of the first phase of the algorithm which can be bounded as in inequality 3.7.

Chapter 4

Implementation

In this project we also provide an implementation of LCD algorithm. We make the implementation available in R through package called *cglearn*. R is broadly used, both in academia and in business, open-source statistical programming language. The name of package is short for chain graph structure learning. It is also analogy to the R package *bnlearn* for learning bayesian networks. The *cglearn* package could be installed through Github and soon will be available on CRAN. Details about package installation are in subsection 4.2.

The package provides functionality for learning chain graphs structure within LCD algorithm and performe conditional independence analysis for given data set. The functionality is further described in subsection 4.1.

It is worth to mention that current implementation of LCD algorithm provides functionality of learning chain graphs only for numeric categorical random variables. The major part of package was implemented in C++ and injected in R using Rcpp package. It was done in purpose of maximizing performance of algorithm. The main part of LCD algorithm implementation took over two thousands lines of code. The source code can be found on GitHub repository at [8].

4.1. Package Functionality

In this section we describe the most important functionalities of *cglearn* package. In the following subsections we will give short description of the package's function, definition of arguments and the result. At the end of this section we suggest possible further improvements of the *cglearn* package.

4.1.1. cglearnLCD

The function *cglearnLCD* is the main function of the package. It performes LCD algorithm for the following given arguments.

- **data** - *data.frame* object with categorical numeric variables.
- **separationTree** - *list* of vectors column ids which defines separation tree based on given data.
- **pValueThr** - *numeric* number from interval $(0, 1)$ which represents p-value threshold to determine conditional independence.

The *cglearnLCD* function returns a *data.frame* object which represents incidence matrix of chaing graph obtained by LCD algorithm.

4.1.2. calcCondIndepend

The function *calcCondIndepend* performs statistical test for conditional independence $x \perp\!\!\!\perp y \mid Z$. Required arguments are listed below.

- **x** - *numeric* categorical vector.
- **y** - *numeric* categorical vector.
- **listOfNumVect** - *list* of categorical numeric vectors.

The *calcCondIndepend* function returns a S3 object which contains value of statistic, number of degrees of freedom and p-value of the test.

4.2. Package Installation

The source code of the implementation of *cglearn* package is placed on GitHub. Any R package which is located in GitHub can be installed by using function *install_github* from package *devtools*. The following listing contains R code for installing *cglearn* package.

```r
if (!require(devtools)) {
    install.packages("devtools")
}
library(devtools)

devtools::install_github("DSkrzypiec/cglearn")
```

Listing 4.1: R code for installing *cglearn* package.

The only dependency of *cglearn* package is package *Rcpp* which binds R and C++ language. The Rcpp package is very common in the world of R packages. Thus such a dependency is not very restrictive. Such a short list of dependent packages makes *cglearn* package easy to install, reliable and stable in further development.

Chapter 5

Results

In this section we present example of usage *cglearn* package. Listing 5.1 contains R code defining data set (*data.frame* in R) and executing LCD algorithm implemented in *cglearn* package. In the first part of listing 5.1 (lines 1-8) we define a mock data set. This approach guarantees easiness of the reproduction of result and furthermore in this way we can confront our perception of conditional structure of defined data set with outcome of LCD algorithm. In lines 10-12 of listing we use function from *cglearn* package to execute LCD algorithm for earlier defined data set, separation tree build only from one node which contains all of the variables and p-value threshold value set to 0.10.

```
1  set.seed(seed = 299938)
2  mockDataFrame <- data.frame(
3      A = c(rep(1, 100), rep(0, 100)),
4      B = c(rep(1, 80), rep(0, 120)),
5      C = c(rep(1, 140), rep(0, 60)),
6      D = c(rep(1, 180), rep(0, 20)),
7      E = sample(x = c(0, 1), size = 200, replace = TRUE)
8  )
9
10 resultIndMatrix <- cglearn::cglearnLCD(data = mockDataFrame,
11                                 separationTree = list(1:5),
12                                 pValueThr = 0.10)
13 resultIndMatrix
```

Listing 5.1: Example usage of *cglearn* package.

Listing 5.2 presents raw output in R console after execution of listing 5.1. The output shows incidence matrix of chain graph obtained by LCD algorithm. Values in the matrix have the following meaning.

- Value 2 represents undirected edge
- Value 1 represents directed edge
- Value 0 represents absence of edge between vertices

The outcome of applying LCD algorithm to obtained chain graph seems to be consistent with common sense of dependence structure of data set defined in the listing 5.1. Chain graph obtained by execution of listing 5.1 is visualized in figure 5.1.

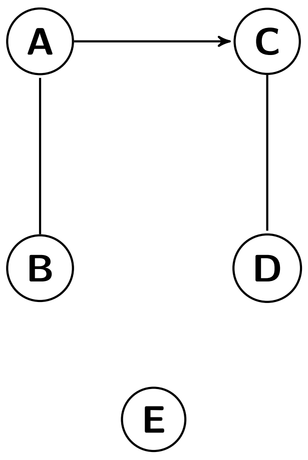

Figure 5.1: Visualization of result chain graph from listing 5.1

```
    A B C D E
  A 2 2 1 0 0
  B 2 2 0 0 0
  C 0 0 2 2 0
  D 0 0 2 2 0
  E 0 0 0 0 2
```

Listing 5.2: Result of execution listing 5.1

Chapter 6

Summary

In this project we have focused on chain graphs and structural learning algorithms. We have presented in details LCD algorithm and the main concept behind the algorithm - separation trees. We described mathematical basis for this algorithm and discussed importance of lowering computational complexity of structural learning algorithms. We have provided efficient implementation of the algorithm which can be accessed by R package *cglearn*.

Bibliography

[1] M. Frydenberg, *The Chain Graph Markov Property*, Scandinavian Journal of Statistics 17 (1990) 333-353

[2] Z. Ma, X. Xie and Z. Geng, *Structural Learning Of Chain Graphs via Decomposition*, Journal of Machine Learning Research 9 (2008) 2847-2880

[3] J. Nielsen, J. Pena and D. Sonntag, *An inclusion optimal algorithm for chain graph structure learning*, Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics, PMLR (2014) 33:778-786

[4] J. Pena, *Faithfulness in Chain Graphs: The Discrete Case*, International Journal of Approximate Reasoning 50 (2009) 1306-1313

[5] J. Pena, *Faithfulness in Cahin Graphs: The Gaussian Case*, Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (2011) 588-599

[6] A. Roverto and L. La Rocca, *On Block Ordering of Variables in Graphical Modelling*, Scandinavian Journal of Statistics 33 (2006) 65-81

[7] M. Studeny and R.R. Bouckaert, *On chain graph models for description of conditional independence structures*, Annals Of Statistics 26 (1998) 1434-1495

[8] GitHub repository containing source code of *cglearn* package, `https://github.com/DSkrzypiec/cglearn` (03.12.2017)