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

April 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 place will be abstract of this project.

Keywords

graphical model, chain graph, structure learning

Thesis domain (Socrates-Erasmus subject area codes)

11.2 Statistics

Subject classification

62 Statistics
62C10 Bayesian Problems

Title of the thesis in Polish

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 where PGM is represented by directed acyclic graph and Markov Fields where PGM is represented by undirected graph. Chain graphs is a class of graphs that does not contains cycles (formal definition in 2.1.10). It contains both directed and undirected edges in graph representation hence it is natural generalization of Bayesian Networks and Markov Fields. Such a generalization was needed because of limitation of Markov Fields and Bayesian Networks. 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-effect relationships without possibility of existence of mutual correlation between two random variables. [TO BE CHANGED] In this paper we present one algorithm for learning chain graphs and one algorithm for learning undirected graphical models. Both algorithms are based on idea of graph decomposition which suppose to decrease complexity of algorithms. [/TO BE CHANGED]

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 $u \rightarrow v$.

Definition 2.1.3. (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 2.1.4. (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 2.1.5. (Route)
A *route* in graph $G = (V, E)$ is a sequence of vertices $(v_0, \ldots, v_k)$, $k \geq 0$, such that

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

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 $u \mapsto v$.

Definition 2.1.6. (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 2.1.7. (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 $\{v_2, v_3, \dots, v_{k-1}\}$ 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)

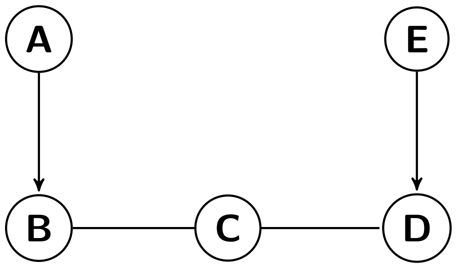

Figure 2.1: Immorality in chain graph

Definition 2.1.8. (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 2.1.9. (Cycle)
A route $r = (v_0, v_1, \dots, v_k)$ 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 2.1.10. (Chain graph)
A graph $G = (V, E)$ is called a chain graph if it does not have directed (pseudo) cycles.

Definition 2.1.11. (Section)
A subroute $\sigma = (v_i, \dots, v_j)$ of route $\rho = (v_0, \dots, v_k)$ in graph G is called section if σ is the maximal undirected subroute of route ρ. That means $v_i - \dots - v_j$ 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 $v_{i-1} \rightarrow v_i$ 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 *head-to-head* section. Otherwise the section is called *non head-to-head.* For a given set of vertices $S \subset V$ in graph G and section $\sigma = (v_i, \dots, v_j)$ we say that section is hit by S if $\{v_i, \dots, v_j\} \cap S \neq \emptyset$. Otherwise we say that section σ is outside set S.

Definition 2.1.12. (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 head-to-head with respect to ρ and σ is outside of S.

2. Section σ is non head-to-head with respect to ρ and σ is hit by S.

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.

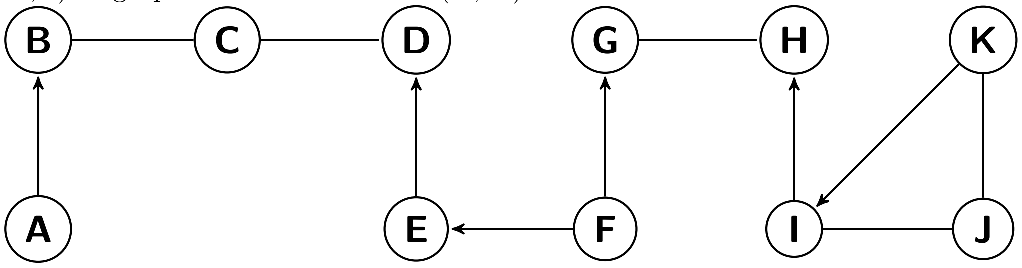

Figure 2.2: Example graph

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.

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 $(X_1, X_2, \ldots, X_n)$ 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 $A_1, A_2, A_3 \in \mathcal{F}$

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

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

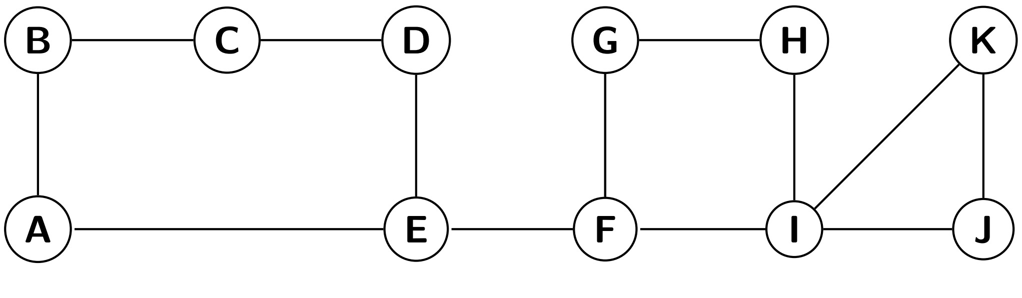

Figure 2.3: Moral graph of graph in figure 2.2

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 [4]. 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

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

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.

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 equivalent 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* [3]. LCD algorithm returns representative chain graph of it's Markov equivalence class, because even with perfect knowledge on the data probability distribution any two chain graph structures within the same Markov equivalence class are indistinguishable. (TODO: Add ref) The algorithm is build from 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.

3.1. Decomposition of chain graphs

3.1.1. Separation Trees

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

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

We will be using graphical convention for representing node trees proposed by Ma, Xie and Geng in [3]. 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.

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

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 $i \in \{1, 2\}$.

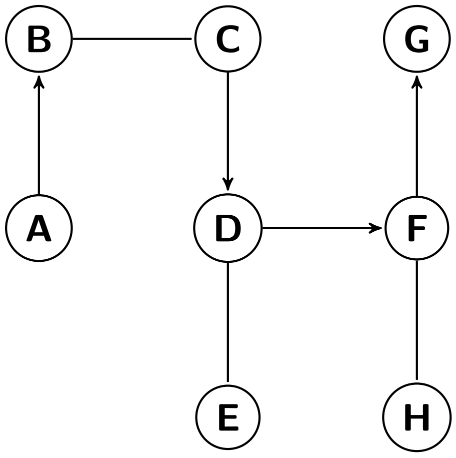

Figure 3.1: Example graph for node tree illustration

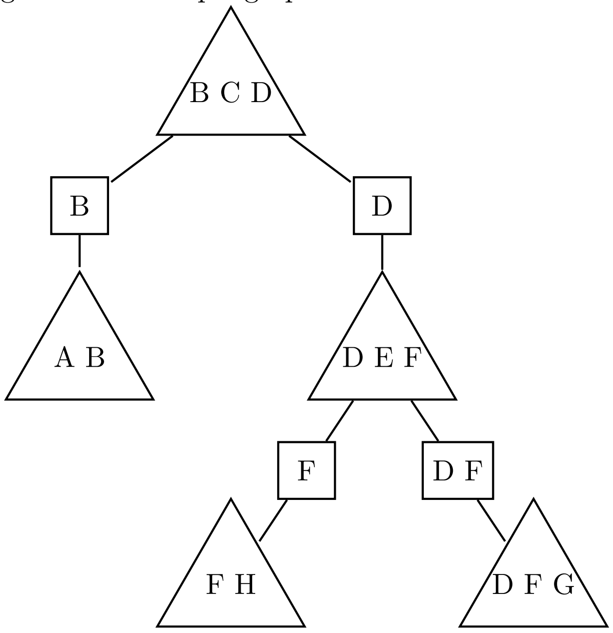

Figure 3.2: Node tree based on graph 3.1 and $\mathcal{C}$

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 $\mathcal{T}(G, \mathcal{C})$ is a separation tree of graph presented in figure 3.1. The node tree $\mathcal{T}(G, \mathcal{C})$ contains four separators $\{B\}$, $\{D\}$, $\{F\}$, and $\{D, F\}$. If we remove a separator $\{D\}$ from $\mathcal{T}(G, \mathcal{C})$ we got $V_1(\{D\}) = \{A, B, C, D\}$ and $V_2(\{D\}) = \{D, E, F, G, H\}$. Condition

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

holds, because every path from $\{A, B, C\}$ to $\{E, F, G, H\}$ is hit by separator $\{D\}$ in graph G. Similar situation appears for every mentioned separator in node tree $\mathcal{T}(G, \mathcal{C})$, so it is actual a separation tree of chain graph G presented in figure 3.1.

3.1.2. Construction of Separation Trees

Some info about constructing separation trees.

3.2. Algorithm

3.2.1. Mathematical basis

There will be mathematical basis for LCD algorithm. In particular theorem 3 from [3].

3.2.2. Skeleton recovery

Some info about SKELETON RECOVERY ALG.

Algorithm 1 (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 $(G_h)_{i \in \{1,...,H}$ 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
```

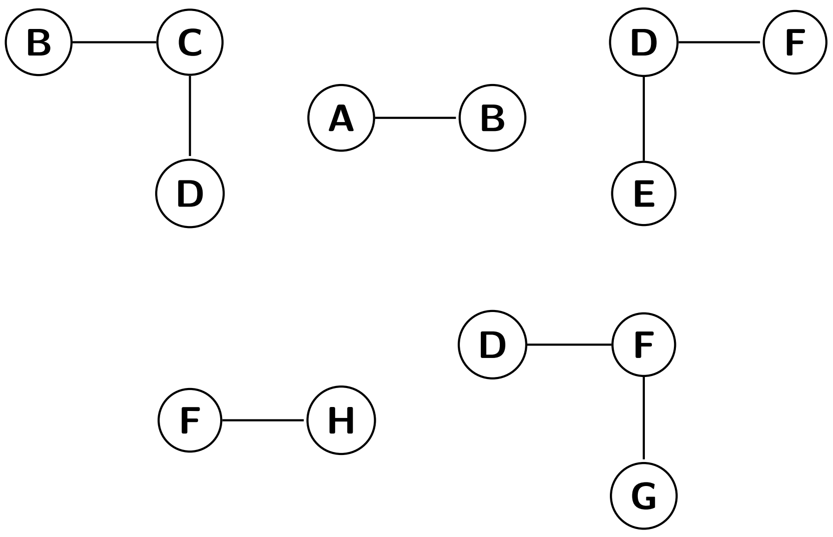

Figure 3.3: Result of first phase of LCD algorithm

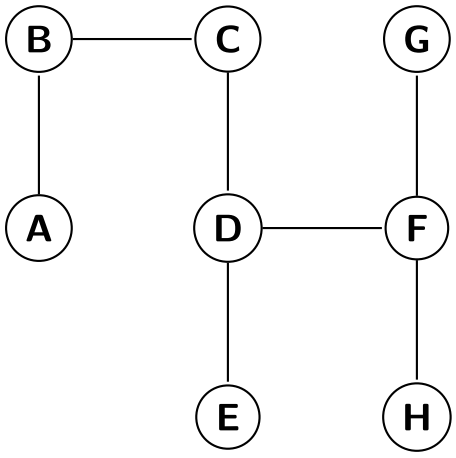

Figure 3.4: Result of second phase of LCD algorithm

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 3.3. 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 3.4. Result of applying skeleton recovery algorithm is the same as outcome from second phase of the algorithm, because there isn't pair of random variable satisfying condition from 20th line of Algorithm 1.

3.2.3. Complexes recovery

Some info about COMPLEX RECOVERY ALG.

Algorithm 2 (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 1.
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
```

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 separation tree, 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 1. 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 1 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

$$\begin{aligned} T(\mathcal{T}) &= \mathcal{O}\left(\sum_{C_i \in H} \frac{1}{2} c_i (c_i + 1) 2^{c_i}\right) \leq \\ &\leq \mathcal{O}\left(\sum_{C_i \in H} \frac{1}{2} m (m + 1) 2^{m}\right) = \\ &= \mathcal{O}\left(k m^2 2^m\right) \end{aligned} \tag{3.2}$$

Chapter 4

ASP Algorithm

Bibliography

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

[2] R. D. Nowak and D. Vats, *A Junction Tree Framework for Undirected Graphical Model Selection*, Journal of Machine Learning Research 15 (2014) 147-191

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

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