HAL
open science

Graph Searches with Applications to Cocomparability Graphs

Jérémie Dusart

To cite this version:

Jérémie Dusart. Graph Searches with Applications to Cocomparability Graphs. Computer Science [cs]. Université Denis Diderot Paris 7, 2014. English. NNT : . tel-01273352

HAL Id: tel-01273352

https://inria.hal.science/tel-01273352v1

Submitted on 12 Feb 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITÉ PARIS.
DIDEROT (Paris 7)
SORBONNE PARIS
CITÉ

ÉCOLE DOCTORALE: 386 – Sciences Mathématiques de Paris Centre
Laboratoire d'Informatique Algorithmique: Fondements et Applications (LIAFA)

DOCTORAT
INFORMATIQUE

DUSART Jérémie

Graph Searches with Applications to Cocomparability Graphs

Parcours de graphes, applications sur les graphes de cocomparabilité

Thèse co-dirigée par Michel Habib, Professeur (Université Paris Diderot) et Derek Corneil, Professeur (University of Toronto)

Soutenue le 24 juin 2014

JURY

Directeurs de thèse:	Michel Habib
	Derek Corneil
Rapporteurs:	Lhouari Nourine
	Christophe Paul
Examinateurs:	Ekkehard Köhler
	Maurice Pouzet
	Laurent Viennot

Acknowledgements

I would like to thank my supervisors Michel and Derek. I am grateful for their supports, advices, discussions, help with the English (especially Derek on this point) and letting me pursue research in an independent way. I also would like to thank Derek for its invitations to Toronto, a city where I met many people who had or going to have a big influence on me.

I wish to thank Christophe Paul and Lhouari Nourine for reading my thesis and writing a report. I don't forget also Maurice Pouzet, Laurent Viennot and Ekkehard Köhler for accepting to be part of the jury.

I am also happy to thank all the other people I met during my master internship and my phd in the LIAFA: the researchers, teachers, staff and of course the other phd students. I had a lot of fun moments there.

Finally, I will also thank my family and friends outside the LIAFA.

Contents

Introduction **1**
0.1 Content of the thesis 2
0.2 Contributions 3

1 Graph Search in cocomparability graphs **5**
1.1 Introduction 5
1.2 Graph Search Paradigm 5
1.3 Classical graph searches 7
1.3.1 Generic Search (GENS) 7
1.3.2 Breadth-First Search (BFS) 8
1.3.3 Depth-First Search (DFS) 9
1.3.4 Lexicographic Breadth-First Search (LBFS) 9
1.3.5 Lexicographic Depth-First Search (LDFS) 11
1.3.6 Maximal Neighborhood Search (MNS) 12
1.3.7 Maximal Cardinality Search (MCS) 13
1.3.8 Hierarchy of graph searches 13
1.4 Comparability, cocomparability and interval graphs 14
1.4.1 Cocomparability and comparability 14
1.4.2 Interval graphs 16
1.5 Graph search in cocomparability graphs 17
1.5.1 Controlling tie-breaking via "+" sweeps 17

1.5.2 Graph searches that maintain a cocomp ordering when used as a "+" sweep . . . 18
1.6 Consequences . . . 21

2 A new characterization theorem for Cocomparability Graphs 23
2.1 Introduction . . . 23
2.2 The maximal antichain lattice of a partial order (MA(P)) . . . 23
2.3 Extra properties of MA(P) . . . 26
2.4 Cliques structure of cocomparability graphs . . . 28
2.5 Conclusion and perspectives for further work . . . 33

3 Interval subgraphs of cocomparability graphs 35
3.1 Introduction . . . 35
3.2 Spanning interval graphs . . . 36
3.3 Computing a maximal chain of MA(P) . . . 40
3.4 Maximal chordal and interval subgraphs . . . 47
3.5 Maximum interval subgraph . . . 52
3.6 Conclusions and perspectives for further work . . . 53

4 Simplicial vertices and clique separators in cocomparability graphs 55
4.1 Introduction . . . 55
4.2 Fully comparable clique and MNS ordering . . . 56
4.3 Simplicial vertices in cocomparability graphs . . . 58
4.4 Minimal clique separators in cocomparability graphs . . . 60
4.4.1 The problem of minimal clique separators . . . 60
4.4.2 Clique separators and components in cocomparability graphs 62
4.4.3 Structure of the atoms of a cocomparability graph . . . 67
4.4.4 Correctness of the Algorithm . . . 70
4.5 Conclusions and perspectives for further work . . . 73

5 Transitive Orientation: Multisweep LBFS 75
5.1 Introduction . . . 75
5.2 Series of LBFS . . . 76
5.3 Proof of Repeated LBFS$^+$ Algorithm . . . 77

5.3.1 Sketch of the proof of our main result 77
5.3.2 Definitions and notations 78
5.3.3 The proof 78
5.4 Conclusions and perspectives for further work 90

6 TBLS: a new model for graph search 93
6.1 Introduction 93
6.2 TBLS, a Tie-Breaking Label Search 94
6.3 Characterizing classical searches using TBLS 96
6.3.1 Generic Search 97
6.3.2 BFS (Breadth-First Search) 98
6.3.3 DFS (Depth First Search) 99
6.3.4 LBFS (Lexicographic Breadth First Search) 99
6.3.5 LDFS (Lexicographic Depth First Search) 100
6.3.6 Limitations of the TBLS model 101
6.4 Two new lexicographic searches 102
6.4.1 LexUP 102
6.4.2 LexDOWN 104
6.5 TBLS hierarchy of searches 106
6.6 The relationship between GLS and TBLS 108
6.7 Test, certification and implementation 111
6.7.1 Testing a TBLS 111
6.7.2 Certifying that an ordering is produced by a given search . 112
6.7.3 Implementation 113
6.8 Application: the TBLS dimension, a new graph parameter 115
6.9 Concluding remarks 116

A Notation 119
A.1 Sets and binary relations 119
A.2 Graphs 119
A.3 Orders 121

References 125

List of Algorithms **131**

Introduction

A graph search is a mechanism for systematically visiting the vertices of a graph. It has been a fundamental technique in the design of graph algorithms since the early days of computer science. Many of the early search methods were based on Breadth First Search (BFS) or Depth First Search (DFS) and resulted in efficient algorithms for practical problems such as the distance between two vertices, diameter, connectivity, network flows and the recognition of planar graphs see [CLRS01].

Many variants of these searches have been introduced since, providing elegant and simple solutions to many problems. For example, Lexicographic Breadth First Search (LBFS) [RTL76], and its generalization Maximal Neighborhood Search (MNS) [Shi84] were shown to yield simple linear time algorithms for chordal graph recognition. Since then many new applications of LBFS have been presented ranging from recognizing various restricted families of graphs to finding vertices with high eccentricity or to finding the modular decomposition of a given graph, see [Cor04a, COS09].

Even if they were known for quite some time, it is only recently that they have been considered a subject of its own. The first paper to do it must be [CK08]. In it, the authors studied the graph search and characterize the ordering they produce. Doing so, they introduced Lexicographic Depth First Search (LDFS) based on its symmetry with LBFS . A few years after its discovery it was shown that LDFS when applied to cocomparability graphs yields simple and efficient

algorithms for solving various Hamiltonian Path related problems [CDH13, MC12, CDHK]. For a general study of graph search, it must also be cited [KSB11, Kru05] where the authors introduced frameworks to capture the classical graph searches.

The purposed of this thesis is to augment these results.

The basic definitions and notations used in this thesis are quite standard. For the reader unfamiliar with graph or order theories, they can be found in Appendix A. In this thesis, unless otherwise stated, the objects (graph, partial order...) are finite.

0.1 Content of the thesis

In chapter 1, we present a formal definition of graph search, review the principal graph searches and show how they fit in the formalism, make a brief survey of cocomparability graphs and interval graphs, and present some results on graph searches and cocomparability graph. These results were the main motivation of the study of cocomparability graphs from a graph search point of view.

Chapter 2 is dedicated to a new characterization theorem for cocomparability graphs. This theorem states that the set of maximal cliques of a cocomparability graph can be equipped with a lattice structure respecting some properties. This theorem can be seen as an extension of the theorem that states that a graph is an interval graph if and only if there exists a linear ordering on its maximal cliques.

In chapter 3 we develop an algorithm to compute a maximal chain of the lattice structure of a cocomparability graph. This algorithm gives us a way to find a minimal interval extension of a partial order but also a maximal interval and maximal chordal subgraph in $O(n^2)$ in a very simple fashion. With this algorithm we can see that graph search can be of great help to design simple algorithms. At the end of the chapter, we deal with the problem of computing a maximum interval subgraph.

Chapter 4 is dedicated to solving the problem of finding the simplicial vertices of a cocomparability graph and the problem of finding a decomposition of a cocomparability graph using the clique separators. In both cases, we present a linear time algorithm to solve the problem. So far these case are the only algorithms known for these problems that run in linear time for a graph class

different from the chordal graph class. A clique separator based decomposition allows us to use a divide and conquer approach for problems like maximum clique or optimal coloring. This result implies a new way to decompose a partial order with good algorithmic perspectives.

In chapter 5, we present an algorithm to find a cocomp ordering or equivalently a transitive orientation. This algorithm answers positively a question asked in [COS09], namely whether there exists a multisweep algorithm based on LBFS to find a cocomp ordering of a cocomparability graph.

Chapter 6 is dedicated to the presentation of a framework powerful enough to capture many graph searches either used individually or in a multi-sweep fashion and simple enough to allow general theorems on graph searches. Building on the General Label Search (GLS) framework from [KSB11] we not only simplify their model but also unify their model with the "pattern-conditions" formalism of [CK08].

0.2 Contributions

- Chapter 1 contains the presentation of a variation of an original work done by Derek Corneil, Michel Habib and Ekkehard Kőhler. Chapters 2, 3, 4 contain work done in collaboration with Michel Habib. The contents of those chapters forms the subject of an article in preparation [CDHK].
- Chapter 5 contains work done with Michel Habib and submitted [DH].
- Chapter 6 contains work done in collaboration with Derek Corneil, Michel Habib, Antoine Mamcarz, Fabien de Montgolfier and submitted [CDH$^+$].

1

Graph Search in cocomparability graphs

1.1 Introduction

Since our work deals with graph search, we start by formalizing what we mean by a graph search. It will be done through the introduction of the Graph Search Paradigm in section 1.2. In section 1.3, we review some classical graph searches and show how they fit in the Graph Search Paradigm. We use the opportunity to recall some of the important results about the graph searches that interest us. In section 1.4, we introduce the comparability, cocomparability and interval graphs. In section 1.5, we present results about graph searches in cocomparability graphs. These results constitute an important basis for the applications in cocomparability graphs and were the main motivation of the study of cocomparability graphs from a graph search point of view.

This chapter is dedicated to the presentation of a variation of a work done by Derek Corneil, Michel Habib and Ekkehard Kőhler.

1.2 Graph Search Paradigm

A graph search "visits" the vertices of the given graph in a vertex-by-vertex fashion and the choice of the next vertex to visit is determined by properties of the vertices already visited. To formalize such an algorithm (presented below as the Graph Search Paradigm) we impose the following conditions:

Conditions for the Graph Search Paradigm (GSP):

1. At any time during the search we let X denote the set of currently visited vertices; initially X is the null set. For notational convenience, $V_{unvisited} = V(G) - X$ denotes the currently unvisited vertices.

2. During the execution of the algorithm, we denote the set of vertices eligible to be visited next as *Eligible*; initially *Eligible* is $V(G)$. We put the condition that for all $u, v \in V_{unvisited}$, if $N(u) \cap X = N(v) \cap X$ then either u and v are both in *Eligible* or they are both not in *Eligible* (i.e., they are both in $V_{unvisited} - Eligible$). We called this condition *Left Twin Compatible (LTC)*.

3. For all steps of the algorithm, we require $Eligible \neq \emptyset$.

It should be noted that we do not put any condition on how the set *Eligible* is computed and what are the informations, data used. It is not necessarily a function of X. We only want the LTC condition to be respected. The LTC condition can be seen as a fairness condition, i.e. two vertices similar from the point of view of the visited vertices cannot be distinguished.

It should be noted that we do not put any condition on the connectivity of the graph. The graph can be disconnected. Condition 3 implies that the search knows what to do in the case of a disconnected graph.

The third condition immediately yields the following property:

Proposition 1.2.1. *Every graph search $\mathcal{S}$ satisfying the Graph Search Paradigm when applied to a graph outputs an ordering $\mathcal{S}(\sigma)$ of the vertices.*

Although the description of how *Eligible* is computed is enough to specify the particular search, many graph searches are closely identified with the data structures that are used in their implementations. To acknowledge this relationship, in our Graph Search Paradigm we introduce a data-structure D used by the algorithm as D and let $D(X)$ represent the status of D at the stage when X is the set of visited vertices. We require that $D(X)$ stores a subset of $V_{unvisited}$ and that all vertices in *Eligible* must be present in $D(X)$. Furthermore note that the Graph Search Paradigm is silent on how the first line of the loop "chooses" v from *Eligible* when $|Eligible| > 1$. In such a case "tie-breaking" is set to have

occurred. At this point, we assume that every vertex in *Eligible* is eligible to be chosen. In subsection 1.5.1 we will describe a more sophisticated "tie-breaking" mechanism.

Algorithm 1: Graph Search Paradigm (GSP).

Input: A graph $G = (V(G), E(G))$
Output: an ordering σ of $V(G)$ where $\sigma(i)$ is the i'th vertex visited
$X \leftarrow \emptyset$ %{X is the set of visited vertices}%;
$V_{unvisited} \leftarrow V(G)$ %{$V_{unvisited}$ is the set of unvisited vertices}%;
$Eligible \leftarrow V(G)$ %{all vertices are eligible to be visited first}%;
Initialize data structure $D(\emptyset)$;
for $i = 1$ *to* n **do**
  Compute *Eligible* from $D(X)$ and choose v from *Eligible* to be the next visited vertex;
  $\sigma(i) \leftarrow v$;
  $V_{unvisited} \leftarrow V_{unvisited} - \{v\}$;
  $X \leftarrow X \cup \{v\}$;
  Update $D(X)$;
end

1.3 Classical graph searches

To illustrate the Graph Search Paradigm (GSP), we review the classical searches Generic Search (GENS), Breadth-First Search (BFS) and Depth-First Search(DFS), their "lexicographic" variants LBFS and LDFS , Maximal Neighborhood Search (MNS) and Maximal Cardinality Search (MCS).

1.3.1 Generic Search (GENS)

A Generic Search as described by Tarjan [Tar72] is any search that wherever possible visits a neighbor of an already visited vertex. Therefore, at any step $Eligible = \{x \in V_{unvisited} \mid \exists v \in X \text{ and } x \in N(v)\}$. This describes the generic

search in the case of a connected graph. To deal with disconnected graphs, we have to put the extra condition that if $Eligible = \emptyset$ then $Eligible = V_{unvisited}$.

There exists a characterization of a generic search ordering:

Theorem 1.3.1. *[CK08] Let $G = (V(G), E(G))$ be a graph and σ an ordering of $V(G)$. The following conditions are equivalent:*

1. *σ is a generic search ordering of $V(G)$.*
2. *For every triple of vertices $a,\ b,\ c$ such that $a <_\sigma b <_\sigma c$, $a \in N(c) - N(b)$ there exists $d \in N(b)$ such that $d <_\sigma b$.*

1.3.2 Breadth-First Search (BFS)

We now focus on the well-known Breadth-First Search (BFS). BFS is a restriction of Generic Search in the sense that BFS will always choose a vertex with a visited neighbor wherever is possible. Therefore every BFS ordering is a generic search ordering. But the converse is not true. BFS is more picky on the set *Eligible*. *Eligible* is the neighborhood in $V_{unvisited}$ of the earliest visited vertex in X having a neighbor in $V_{unvisited}$. To handle the case of disconnected graphs, again we set the condition that in the case where every vertex of X does not have a neighbor in $V_{unvisited}$, then $Eligible = V_{unvisited}$.

A characterization of a BFS ordering exists:

Theorem 1.3.2. *[CK08] Let $G = (V(G), E(G))$ be a graph and σ an ordering of $V(G)$. The following conditions are equivalent:*

1. *σ is a BFS ordering .*
2. *for every triple $a, b, c \in V$ such that $a <_\sigma b <_\sigma c$, $a \in N(c) - N(b)$, there exists d such that $d \in N(b)$ and $d <_\sigma a$.*

To implement BFS, the most common way is by encoding $D(X)$ using a priority queue of vertices (see [CLRS01]). This way we get a complexity $O(n+m)$.

1.3.3 Depth-First Search (DFS)

We now turn our attention to Depth-First Search (DFS). DFS, as BFS, is also a restriction of Generic Search. The difference between DFS and Generic Search lies again in the computation of *Eligible*. *Eligible* is the neighborhood in $V_{unvisited}$ of the latest visited vertex of X having a neighbor in $V_{unvisited}$. To handle the case of disconnected graphs, again we set the condition that in the case of every vertex of X not having a neighbor in $V_{unvisited}$, then $Eligible = V_{unvisited}$.

A DFS ordering can be characterized:

Theorem 1.3.3. *[CK08] Let $G = (V(G), E(G))$ be a graph and σ an ordering of $V(G)$. The following conditions are equivalent:*

1. *σ is a DFS ordering.*
2. *for every triple of vertices $a,\ b,\ c$ such that $a <_\sigma b <_\sigma c$, $a \in N(c) - N(b)$ there exists $d \in N(b)$ such that $a <_\sigma d <_\sigma b$.*

To implement DFS, the most efficient way is by encoding $D(X)$ using a stack (see [CLRS01]). This way we get a complexity $O(n + m)$.

1.3.4 Lexicographic Breadth-First Search (LBFS)

The Lexicographic Breadth-First Search (LBFS or LexBFS) was first introduced in [RTL76] to recognize chordal graphs. Since then, many new applications of LBFS have been presented ranging from recognizing various restricted families of graphs to finding vertices with high eccentricity or to finding a modular decomposition of a given graph, see [Cor04a, COS09, Cor04b, BCHP08, TCHP08, LW13, DH]. The LBFS algorithm assigns label to each vertex that are words over the alphabet $\{0, ..., n-1, n\}$ and the empty word is represented by ϵ. At each step, a vertex with lexicographic largest label is chosen and the label of its neighbors is updated. Therefore, *Eligible* is the subset of $V_{unvisited}$ having the lexicographic largest label.

Let us now describe LBFS by an algorithm and then give an example. It must be noted that we number the vertices from 1 to n whereas when it was first described in [RTL76], the vertices where numbered from n to 1.

Algorithm 2: LBFS

Data: $G = (V(G), E(G))$ an undirected graph
Result: an ordering σ of the vertices of G

```
assign label ϵ to all vertices;
for (i = 1 to |V(G)|) do
    pick any unvisited vertex u with lexicographically largest label;
    σ(i) ← u % {give to u the number i}%;
    foreach (unvisited vertex v ∈ N(u)) do
        append |V(G)| − i to label(v);
    end
end
Output σ;
```

To illustrate LBFS, consider the graph in Figure 1.1. The value of the labels at the beginning of each step is in the table of Figure 1.2. The start vertex is d and the ordering produced is $\sigma =< d, c, b, f, a, e >$.

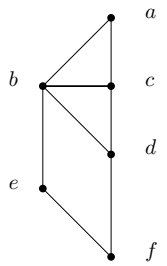


Figure 1.1: A graph $G = (V(G), E(G))$.

A characterization of a LBFS ordering exists:

Theorem 1.3.4. *[RTL76, Gol04, BDN97] Let $G = (V(G), E(G))$ be a graph and σ an ordering of $V(G)$. The following conditions are equivalent:*

1. *σ is a LBFS ordering*
2. *for every triple $a, b, c \in V(G)$ such that $a <_\sigma b <_\sigma c$, $a \in N(c) - N(b)$, there exists $d <_\sigma a$, $d \in N(b) - N(c)$.*

vertex	i=1	i=2	i=3	i=4	i=5	i=6
d	ϵ					
c	ϵ	5				
b	ϵ	5	54			
f	ϵ	5	5	5		
a	ϵ	ϵ	4	43	43	
e	ϵ	ϵ	ϵ	3	32	32

Figure 1.2: The value of the labels at the beginning of each step on example 1.1.

The easiest implementation uses partition refinement for the management of $D(X)$ (see [HMPV00]) and has complexity $O(n+m)$.

1.3.5 Lexicographic Depth-First Search (LDFS)

The Lexicographic Depth-First Search (LDFS or LexDFS) was introduced in [CK08]. During quite some time, LDFS has been introduced but no application of it was known. Since then some applications have been found. The most notable ones are to find a maximum path cover of a cocomparability graph [CDH13] and to find a longest path in cocomparability graphs [MC12].

As LBFS , LDFS uses labels that are words over the alphabet $\{0, ..., n-1, n\}$ and the empty word is represented by ϵ. At each step, *Eligible* is the subset of $V_{unvisited}$ having the lexicographic largest label. The difference between the two lies within the way of updating the label of an unvisited neighbor of the chosen vertex. In LBFS , we append $n-i$ whereas in LDFS we prepend i.

Let us now describe LDFS by an algorithm and then give an example.

To illustrate LDFS, consider the graph in Figure 1.1. The value of the labels at the beginning of each step is in the table of Figure 1.3. The start vertex is a and the ordering produced is $\sigma =< a, b, c, d, f, e >$.

A LDFS ordering can be characterized:

Theorem 1.3.5. *[CK08] Let $G = (V(G), E(G))$ be a graph and σ an ordering of $V(G)$. The following conditions are equivalent:*

Algorithm 3: LDFS

Data: $G = (V(G), E(G))$ an undirected graph
Result: an ordering σ of the vertices of G

```
assign label ϵ to all vertices;
for (i = 1 to |V(G)|) do
    pick any unnumbered vertex u with lexicographically highest label;
    σ(i) ← u % {give to u the number i}%;
    foreach (unnumbered vertex v ∈ N(u)) do
        prepend i to label(v);
    end
end
Output σ;
```

vertex	i=1	i=2	i=3	i=4	i=5	i=6
a	ϵ					
b	ϵ	1				
c	ϵ	1	21			
d	ϵ	ϵ	2	32		
f	ϵ	ϵ	ϵ	ϵ	4	
e	ϵ	ϵ	2	2	2	25

Figure 1.3: The value of the labels at the beginning of each step on example 1.1.

1. *σ is a LDFS ordering*

2. *for every triple $a, b, c \in V$ such that $a <_\sigma b <_\sigma c$, $a \in N(c) - N(b)$, there exists $a <_\sigma d <_\sigma b$, $d \in N(b) - N(c)$.*

The best implementation known of LDFS has complexity $O(min(n^2, mloglogn))$ (see [Spi]).

1.3.6 Maximal Neighborhood Search (MNS)

We now turn our attention on Maximal Neighborhood Search (MNS). It has been introduced in [Shi84] for chordal graphs recognition. *Eligible* is the subset

of vertices in $V_{unvisited}$ whose neighborhood in X is maximal with respect to inclusion.

We now recall the characterization of a MNS ordering.

Theorem 1.3.6. *[CK08] The following conditions are equivalent:*

1. *σ is a MNS ordering*

2. *for every triple $a, b, c \in V$ such that $a <_\sigma b <_\sigma c$, $a \in N(c) - N(b)$, there exists $d \in N(b) - N(c)$, $d <_\sigma b$.*

1.3.7 Maximal Cardinality Search (MCS)

To finish with classical searches, we turn our attention on Maximal Cardinality Search (MCS). It has been introduced in [TY84]. *Eligible* is the subset of vertices in $V_{unvisited}$ that have the maximum number of neighbors in X.

MCS can be implemented with complexity $O(n + m)$ (see [TY84]).

1.3.8 Hierarchy of graph searches

A hierarchy exists among the graph searches cited in this section. This hierarchy has first appears in [Shi84] and we summarize it in Figure 1.4. We will come back to this hierarchy in chapter 6.

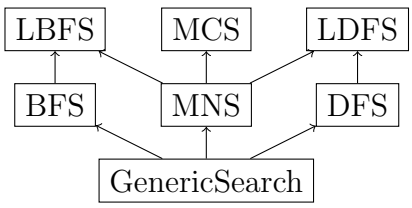


Figure 1.4: Summary of the heredity relationships proved in theorem 6.5.5. An arc from Search S to search S' means that S' extends S.

1.4 Comparability, cocomparability and interval graphs

In this section we make a short review of comparability, cocomparability and interval graphs. As we will show in section 1.5 and later in this thesis, graph searches play an important role in these graph classes. For a more detailed review on graph classes, we refer the reader to [Gol04].

1.4.1 Cocomparability and comparability

Comparability and cocomparability graphs are important both in graph and order theories. To be able to define these families we need first to define the concept of transitive orientation.

Definition 1.4.1. For a graph $G = (V(G), E(G))$, the binary relation $<_o$ over the ground set $V(G)$ is a transitive orientation of G if and only if $<_o$ is transitive and for every $x,\ y \in V(G)$, $xy \in E(G)$ if and only if either $x <_o y$ or $y <_o x$.

Definition 1.4.2. A simple undirected graph $G = (V(G), E(G))$ is a comparability graph if and only if there exists a transitive orientation of G.

A simple undirected graph $G = (V(G), E(G))$ is a cocomparability graph if and only if $\overline{G}$ is a comparability graph.

The greatest interest in comparability graphs comes from the fact that from every partial order we can associate a comparability graph and for every comparability graphs we can associate a partial order. Therefore they naturally arise in problems as soon as there exists a partial order relation. The comparability graph is obtain as soon as we decide to put an edge between two comparable elements and the cocomparability graph when we decide to put an edge between two incomparable element. Examples can be found in Figure 1.5.

Both the comparability and the cocomparability graphs can be characterized by a special ordering of their vertices.

Theorem 1.4.3. *[KS93] $G = (V(G), E(G))$ is a comparability graph if and only if there exists an ordering σ of its vertices such that for $x,\ y,\ z \in V(G)$, $x <_\sigma y <_\sigma z$, $xy, yz \in E(G)$ implies $xz \in E(G)$.*

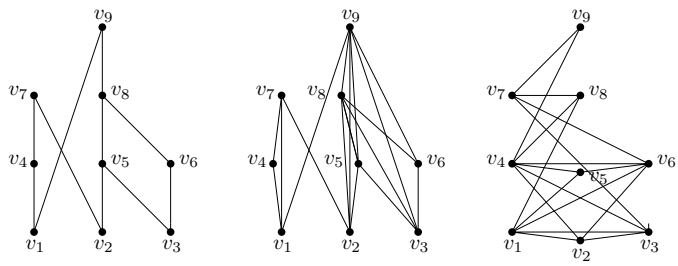


Figure 1.5: From left to right: the directed covering graph of a partial order, the comparability graph and the cocomparability graph associated to this partial order.

We give a proof of the theorem. As we will show, this theorem just states that we can use a linear extension of a transitive orientation of a comparability graph to characterize it.

Proof. Assume that G is a comparability graph and let $(V(G), <_P)$ be a transitive orientation of G. Let σ be a linear extension of $(V, <_P)$. Let $x,\ y,\ z \in V(G)$ such that $x <_\sigma y <_\sigma z$, $xy, yz \in E(G)$. Since σ is a linear extension of P, using the transitivity we deduce that $x <_P z$ and so $xz \in E(G)$.

Conversely, assume that σ is an ordering of $V(G)$ such that for $x,\ y,\ z$, $x <_\sigma y <_\sigma z$, $xy, yz \in E(G)$ implies $xz \in E(G)$ for a graph G. Let (V,$<_P$) be the binary relation defined by for $x,\ y \in V(G)$, $x <_P y$ if and only if $xy \in E(G)$ and $x <_\sigma y$. From the structure of σ, we get that $<_P$ is transitive. Therefore G is a comparability graph. □

The next theorem is in fact the dual version of the previous theorem.

Theorem 1.4.4. *[KS93]*

$G = (V(G), E(G))$ is a cocomparability graph if and only if there exists a ordering σ of its vertices such that for $x,\ y,\ z$, $x <_\sigma y <_\sigma z$, $xz \in E(G)$ then $xy \in E(G)$ or $yz \in E(G)$.

Proof. Assume that G is a cocomparability graph.
$\iff$ $\overline{G}$ is a comparability graph.
$\iff$ there exists an ordering σ of $V(G)$ such that for $x,\ y,\ z \in V(G)$, $x <_\sigma y <_\sigma z$, $xy, yz \notin E(G)$ implies $xz \notin E(G)$.
$\iff$ there exists an ordering σ of $V(G)$ such that for $x,\ y,\ z \in V(G)$, $x <_\sigma y <_\sigma$

z, ($xy \in E(G)$ or $yz \in E(G)$) or $xz \notin E(G)$
$\iff$ there exists an ordering σ of $V(G)$ such that for $x,\ y,\ z \in V(G)$, $x <_\sigma y <_\sigma z$, $xz \in E(G)$ implies $xy \in E(G)$ or $yz \in E(G)$

□

An ordering that satisfies the previous theorem is called a cocomparability ordering or a cocomp ordering for short. For a cocomparability graph $G = (V(G), E(G))$, since a cocomp ordering σ gives us a transitive orientation of the complement, and so a partial order, we will denoted by P_σ the transitive orientation obtained using σ to orient $\overline{G}$.

1.4.2 Interval graphs

Interval graphs form a well-known subclass of cocomparability graphs.

Definition 1.4.5. A graph $G = (V(G), E(G))$ is an interval graph if and only it is the intersection graph of intervals on a line.

An example can be found in Figure 1.6. Interval graphs arise naturally in real-life problem when time management is involved. For examples, see [Gol04].

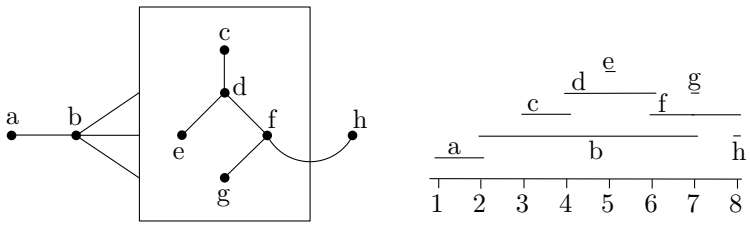


Figure 1.6: An interval graph and its intervals model.

We now review some of the most well-known characterizations of interval graphs.

Theorem 1.4.6. *[Ola91, RR88, Ray87] G is an interval graph if and only if there exists an ordering σ of its vertices such that for $x,\ y,\ z$, $x <_\sigma y <_\sigma z$, $xz \in E(G)$ then $xy \in E(G)$.*

An ordering that satisfies the previous theorem will be called an interval ordering.

If we consider example 1.6, $a,\ b,\ c,\ d,\ e,\ f,\ g,\ h$ is an interval ordering.

Theorem 1.4.7. *[GH64] G is an interval graph if and only if there exists an ordering of its maximal cliques such that for every vertex, the maximal cliques containing it occur consecutively.*

We say that an ordering of cliques satisfies the consecutiveness property if for every vertex, the cliques that contain it occur consecutively.

Again, if we consider example 1.6, $\{a,\ b\}$, $\{b,\ c,\ d\}$, $\{b,\ d,\ e\}$, $\{b,\ d,\ f\}$, $\{b,\ f,\ g\}$, $\{f,\ h\}$ is an ordering that satisfies the previous theorem.

Theorem 1.4.8. *[LC62] G is an interval graph if and only if G is a cocomparability graph and a choral graph.*

We recall that a chordal graph is a graph without induced cycle of length greater or equal than four.

1.5 Graph search in cocomparability graphs

We explained in the first and second section how a graph search $\mathcal{S}$ can give us a total ordering of the vertices of a graph with the property to be a $\mathcal{S}$ ordering. In the third section, we reviewed some graph classes, which have a characterization using an ordering of their vertices. So now, we show that we can use the graph search to get characterizing orderings for cocomparability graphs with the extra property to be a search ordering.

1.5.1 Controlling tie-breaking via "+" sweeps

As mentioned before, the Graph Search Paradigm does not specify how ties are broken when $|Eligible| > 1$. In this subsection we examine how a given total ordering τ of $V(G)$ is often used to break ties when $|Eligible| > 1$. This technique can be used with any given graph search $\mathcal{S}$ where the "+" version is denoted $\mathcal{S}^+(\tau)$. In a "+" sweep the vertex to be chosen is the rightmost *Eligible* vertex as ordered by τ as follows:

The + tie-break rule: Initially the first vertex of $\sigma = \mathcal{S}^+(\tau)$ is chosen to be the rightmost vertex of τ. Thereafter if $|Eligible| > 1$, then the next vertex of σ is the rightmost *Eligible* vertex as ordered by τ.

One advantage of the "+" rule is that it makes the output of a search deterministic. Given a search $\mathcal{S}$, a graph G and an ordering τ, there exists only one ordering σ such that $\sigma = \mathcal{S}^+(G, \tau)$.

Usually τ itself is a vertex ordering produced by a graph search. This technique has been successfully used with LBFS where the initial ordering is an arbitrary LBFS and subsequent LBFS$^+$ sweeps are then employed; for example, to recognize Unit Interval graphs an LBFS followed by two LBFS$^+$ sweeps suffices [Cor04b], whereas to recognize Interval graphs an LBFS, followed by four LBFS$^+$ sweeps, followed by an LBFS that queries the two previous LBFS$^+$ sweeps suffices [COS09].

1.5.2 Graph searches that maintain a cocomp ordering when used as a "+" sweep

We first present a key definition based on a vertex having a private neighbor with respect to another vertex. This notion plays a key role in determining which searches transform a given cocomp ordering into o cocomp ordering that is also that specific search.

Definition 1.5.1. A vertex $y \in Eligible$ satisfies the Private Neighbor Property (PNP) with respect to graph search $\mathcal{S}$ if for every $z \in V_{unvisited} - Eligible$, with $yz \notin E$, there exists an $x \in X$ such that $xy \in E$ and $xz \notin E$.

A search $\mathcal{S}$ satisfies the Private Neighbor Property if at every step of the search (i.e., for every subset X of visited vertices) every vertex in *Eligible* satisfies the PNP.

Theorem 1.5.2. *For a cocomparability graph $G = (V(G), E(G))$, let τ be a cocomp ordering of G. If $\mathcal{S}$ is a graph search satisfying both the Graph Search Paradigm*[1] *and the Private Neighbor Property then $\sigma = \mathcal{S}^+(G, \tau)$ is also a cocomp*

[1] In fact the second condition of the GSP is not required for the proof.

ordering. Furthermore if τ is a linear extension of the partial order $P = (V(G), \leq)$, then σ is a linear extension of P^- (where P^- is the partial order obtained from P by reversing all the arcs).[1]

Proof. To prove this theorem, we need to prove the following fact.

Claim 1.5.3. *For every $uv \notin E(G)$, $u <_\tau v$ if and only if $v <_\sigma u$*

Proof. Suppose it is false, and let us consider $u \neq v \in V(G)$ such that :

1. $u <_\tau v$ and $u <_\sigma v$
2. u is leftmost in σ with such a v.

Let us consider X_u the set of visited vertices when u was chosen by $\mathcal{S}$. At this time v could not be eligible, else it, or an eligible vertex to its right, would have been selected using the + tie-break rule, thereby contradicting the selection of u. So $v \in V_{unvisited} - Eligible$. Since σ is a graph search satisfying the PNP, there exists at least one vertex w such that: $w <_\sigma u$ with $wu \in E(G)$ and $wv \notin E(G)$. By the choice of u being leftmost, necessarily $v <_\tau w$, but then u, v, w is an umbrella in τ contradicting τ being a cocomp ordering. □

To finish the proof of the theorem, let us suppose that σ is not a cocomp ordering, i.e., that σ has an umbrella on three vertices: $a <_\sigma b <_\sigma c$, with $ac \in E(G)$ and $ab, bc \notin E(G)$. Using the above claim, we have: $c <_\tau b <_\tau a$, and therefore τ also is not a cocomp ordering. The second statement of the theorem follows immediately.

□

Before examining which searches produce a cocomp ordering when applied as a "+" search on a given cocomp ordering, we show that by making a slight adjustment, the above theorem can be transformed into a characterization.

Theorem 1.5.4. *Let τ be a cocomparability ordering of the connected graph G and let $\mathcal{S}$ be a search satisfying the Graph Search Paradigm; further, let $\sigma = \mathcal{S}^+(\tau)$. Then σ is a cocomparability ordering if and only if at all stages of $\mathcal{S}^+(\tau)$ the rightmost vertex of Eligible satisfies the PNP.*

[1] This fact is an easy consequence of the definitions of the + tie-break rule and cocomp orderings and will be omitted in the subsequent similar results.

Proof. ($\Leftarrow$) This follows from the previous theorem.

($\Rightarrow$) Assume σ is a cocomparability order, but for some X, u the rightmost vertex of *Eligible* in τ does not satisfy the PNP as witnessed by vertex $v \in V_{unvisited} - Eligible$ where $uv \notin E$ and there is no x in X such that $xu \in E, xv \notin E$. Since $u \in Eligible$ and $v \in V_{unvisited} - Eligible$, by the left twin compatible condition we see that $N(u) \cap X \neq N(v) \cap X$. Thus, since u does not have a private neighbor in X with respect to v, v has a private neighbor $w \in X$ with respect to u.

Now we see that $w <_\sigma u$ since $w \in X$ and $u <_\sigma v$ since v was unvisited when u was chosen. Thus $\{w, u, v\}$ forms an umbrella in σ, contradicting σ being a cocomparability ordering. □

We now look at the question of which searches produce a cocomp ordering when applied as a "+" search on a given cocomp ordering. In light of theorem 1.5.2, it suffices to show that any search that satisfies conditions 1 and 3 of the GSP also satisfies the PNP; in light of theorem 1.5.4 it suffices to show that any search that satisfies the GSP also has the last vertex of every *Eligible* satisfying the PNP.

The following lemma shows that all searches mentioned in section 1.3 satisfy the requirements of theorem 1.5.2.

Lemma 1.5.5. *For all $\mathcal{S}$ in $\{GENS, BFS, DFS, MNS, LBFS, LDFS, MCS\}$, $\mathcal{S}$ satisfies both the Graph Search Paradigm and the Private Neighbor Property.*

Proof. It it clear from their definitions that all searches satisfy the GSP; we deal with PNP satisfaction on a case by case basis.

For generic search, since every vertex with a neighbor in X belongs to *Eligible*, the vertices in $V_{unvisited} - Eligible$ have no neighbor in X. Therefore generic search satisfies the PNP.

Using the usual data structures: a queue (respectively a stack) for BFS (respectively DFS), one can view these searches as selecting at every step, (i.e., for every set $X \subseteq V$ of visited vertices) *Eligible* as the set of unvisited neighbors of some vertex $x \in X$. Therefore no vertex in $V_{unvisited} - Eligible$ is adjacent to x and thus both searches satisfy the PNP.

For all searches that are MNSs (such as LBFS, LDFS, MCS) for all $y \in Eligible$ and for all $z \in V_{unvisited} - Eligible$ we know that $(N(y) \cap X) \nsubseteq (N(z) \cap X)$

and thus for every such y, z pair, y has a private neighbor in X with respect to z.

□

Corollary 1.5.6. *For any cocomp ordering τ and any graph search $\mathcal{S}$ in $\{GENS$, BFS, DFS, MNS, $LBFS$, $LDFS$, MCS, $RMN\}$, $\sigma = \mathcal{S}^+(\tau)$ is also a cocomp ordering.*

Proof. This follows immediately from theorem 1.5.2 and lemma 1.5.5. □

1.6 Consequences

Using the results of the previous sections, we now have a tool-box to produce cocomp orderings with particular properties (inherited from the search we have used). With our toolbox we can produce cocomp orderings which are BFS (resp. DFS, MNS, LDFS or LBFS) orderings. Applications of this will be presented in the rest of the thesis.

It must be noted that for our toolbox to work, we need a cocomp ordering. It is known that given a cocomparability graph, a cocomp ordering can be computed in linear time [MS99]. But this algorithm is quite involved and other algorithms in $O(n + mlogn)$ are easier to handle [HMPV00, MS99]. A new algorithm for finding a cocomp ordering using only LBFS $^+$ will be presented in this thesis in chapter 5.

It should also be noticed that our toolbox can be easily extended to min-LBFS as defined in [Mei05].

So far, in the third section we have introduced interval graph and interval orderings but we do not talk about them in the toolbox. The reason is that an interval ordering is already a {GENS, BFS, DFS, MNS, LBFS, LDFS, MCS, RMN} ordering.

2

A new characterization theorem for Cocomparability Graphs

2.1 Introduction

This chapter is dedicated to a new characterization theorem for cocomparability graphs. This theorem states that the set of maximal cliques of a cocomparability can be equipped with a lattice structure respecting some properties. This theorem can be seen as a generalization of a theorem for interval graphs.

This chapter is organized as follows. In section 2.2, we introduce some terminologies and review the main results about the maximal antichain lattice of a partial order that we need for the proof. In section 2.3 we prove some additional properties about this maximal antichain lattice. Section 2.4 is devoted to the proof of the new characterization theorem.

This work has been done in collaboration with Michel Habib.

2.2 The maximal antichain lattice of a partial order (MA(P))

From a partial order P, we can build different lattices like the Antichain lattice, the Maximal Sized Antichain Lattice... One of them is the lattice of maximal antichains. As proved in [B.88], the set of maximal (under inclusion) antichains of a given partial order P denoted by $MAP(P)$, forms a lattice when equipped

with the following binary relation $\leq_{MA(P)}$: For $A,\ B \in MAP(P)$, $A \leq_{MA(P)} B$ if and only if $\forall y \in B$, $\exists x \in A$ such that $x \leq_P y$. We denote this lattice by $MA(P) = (MAP(P), \leq_{MA(P)})$.

In this section we introduce some terminology and review the main results known about $MA(P) = (MAP(P), \leq_{MA(P)})$, which interest us.

Theorem 2.2.1. *[B.88] Let P be a partial order. $MA(P) = (MAP(P), \leq_{MA(P)})$ is a partial order.*

The next lemma show that the definition is self-dual.

Lemma 2.2.2. *[B.88] Let A, B be two maximal antichains of a partial order P. $A \leq_{MA(P)} B$ if and only if $\forall x \in A$, $\exists y \in B$ with $x \leq_P y$.*

We now introduce some terminology and prove a short proposition. For two maximal antichains $A,\ B$ of a partial order $P = (V(G), \leq_P)$, let $S^-(A, B) = \{x \in A - B | \exists y \in B - A$ with $x <_P y\}$ and $S^+(A, B) = \{x \in A - B | \exists y \in B - A$ with $y <_P x\}$.

Proposition 2.2.3. *Let $A,\ B$ be two maximal antichains of a partial order P. $A = (A \cap B) \cup S^-(A, B) \cup S^+(A, B)$.*

Proof. Let us consider a vertex x of A. We have two cases: either $x \in B$ or $x \notin B$. In the first case $x \in A \cap B$. In the second case, since B is a maximal antichain and $x \notin B$, there must exist $y \in B$, and y is comparable to x. Since $y \in B$ and $x \notin B$, we deduce that $y \neq x$. Therefore we have two cases: either $x <_P y$ or $y <_P x$. In the first case, $x \in S^-(A, B)$ and in the second case $x \in S^+(A, B)$. Thereby $A = (A \cap B) \cup S^-(A, B) \cup S^+(A, B)$. □

For a partial order $P = (X, \leq_P)$, $S \subseteq X$, $M^+(S) = \{v \in S \mid \forall u \in S,\ u \leq_P v$ or $u \parallel_P v\}$. $M^+(S)$ can be seen as the set of sinks of the partial order induced by S. In the same way $M^-(S) = \{v \in S \mid \forall u \in S,\ v \leq_P u$ or $u \parallel_P v\}$. $M^-(S)$ can be seen as the set of sources of the partial order induced by S. Inc(S)=$\{x \in V(G) - S \mid \forall y \in S,\ x \parallel_P y\}$.

For two maximal antichains $A,\ B$ of a partial order P, we define the binary operators $A \wedge_{MA(P)} B = (A \cap B) \cup S^-(A, B) \cup S^-(B, A) \cup M^+(Inc((A \cap B) \cup$

$S^-(A,B) \cup S^-(B,A)))$ and $A \vee_{MA(P)} B = (A \cap B) \cup S^+(A,B) \cup S^+(B,A) \cup M^-(Inc((A \cap B) \cup S^+(A,B) \cup S^+(B,A)))$.

What is important to keep in mind is that we will find in the infimum $A \cap B$, one part of the vertices of A, one part of the vertices of B plus some vertex and in the supremum we will also find $A \cap B$, all the vertices of $A \cup B$ that we did not put in the infimum plus some vertex. Examples can be found in figure 2.2 and in figure 2.3.

Theorem 2.2.4. *[B.88] Let P be a partial order.*
$MA(P) = (MAP(P), \wedge_{MA(P)}, \vee_{MA(P)})$ is a lattice.

Let now establish the structure of $MA(P)$ in the case of an interval order. We recall that an interval order is a partial order without any $2K_2$ (see Figure 2.1).The graph induced by an interval order is an interval graph.

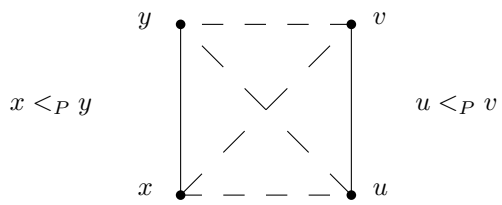


Figure 2.1: A $2K_2$ of a partial order P.

Proposition 2.2.5. *[B.88] P is an interval order if and only if $MA(P)$ is a chain.*

Further questions can be asked about the lattice such that: does $MA(P)$ have a particular structure? What is the maximum size of $MA(P)$ given n?

The answer to the first question is no. Markowsky in [Mar75] and [Mar92] showed that any finite lattice is isomorphic to the galois lattice of a binary relation. This is equivalent to saying that any finite lattice is isomorphic to the maximal antichain of a height one partial order. This result has been rediscovered by Berhendt in [B.88]. This is summarized in the next theorem.

Theorem 2.2.6. *[B.88, Mar75, Mar92] Any finite lattice is isomorphic to the lattice $MA(P)$ of some finite partial order.*

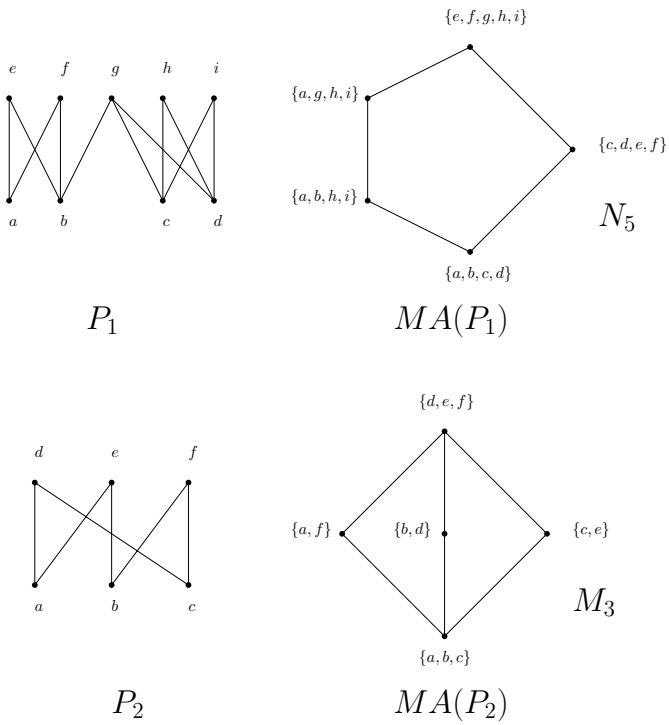


Figure 2.2: Two orders whose maximal antichain lattices are respectively N_5 and M_3.

For the second question, the size of $MA(P)$ can be, unfortunately, exponential in the number of vertices of G. Let us consider the example of Figure 2.3 where there are k chains of 2 vertices resulting in 2^k maximal antichains.

2.3 Extra properties of MA(P)

In this section, we prove some extra properties we need in order to prove the characterization theorem of cocomparability graphs. The first two lemmas are tools that we need to prove the proposition for the characterization theorem.

Lemma 2.3.1. *Let A, B be two different maximal antichains of a partial order P.*

$A <_{MA(P)} B$ if and only if $\forall y \in B - A$, $\exists x \in A - B$ such that $x <_P y$.

Proof. Suppose that $A <_{MA(P)} B$ and let $y \in B - A$. Using the definition of $MA(P)$ on A and B, there exists an element $x \in A$ such that $x \leq_P y$. Since

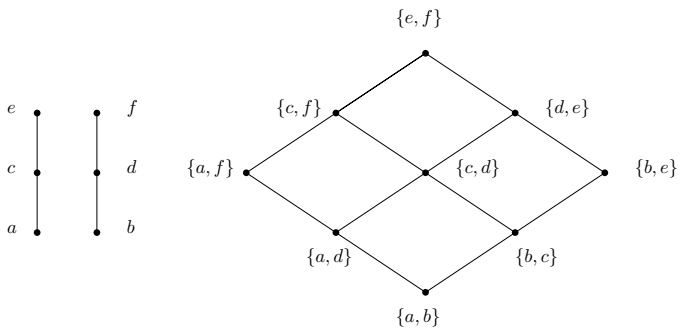


Figure 2.3: An easy extension of this example shows than $MA(P)$ can be of exponential size.

$y \notin A$ we have that $x \neq y$. Moreover B is an antichain and so $x \notin B$. As a consequence, we have that $x \in A - B$ and $x <_P y$.

Conversely, assume that A, B are different maximal antichains of a partial order P and $\forall y \in B - A$, $\exists x \in A - B$ such that $x <_P y$. For all $y \in B \cap A$ we have that $y \leq_P y$. For all $y \in B - A$, using the assumption that $\forall y \in B - A$, $\exists x \in A - B$ such that $x <_P y$, we deduce that $\forall y \in B$, $\exists x \in A$ such that $x \leq_P y$. Hence $A \leq_{MA(P)} B$. Since we assume that $A \neq B$, we conclude that $A <_{MA(P)} B$. □

Lemma 2.3.2. *Let A, B be two maximal antichains of a partial order P such that $A \leq_{MA(P)} B$.*

If $x \in A$, $y \in B$ then $x \leq_P y$ or $x \parallel_P y$.

Proof. We have two cases: either $A = B$ or $A \neq B$. In the first case, since A is an antichain and $A = B$, we get $\forall x \in A$, $\forall y \in A$, if $x \neq y$ then $x \parallel_P y$ and if $x = y$ then $x \leq_P y$. In the second case, suppose for contradiction there exists $x \in A$, $y \in B$ such that $y <_P x$. Since we are in the case where A and B are different maximal antichains, we have $x \in A - B$, $y \in B - A$ and also $A <_{MA(P)} B$. Applying lemma 2.3.1 on A and B, there exists $z \in A - B$ such that $z <_P y$. By transitivity of P, we establish that $z <_P x$, therefore contradicting A being an antichain. □

We now prove the two propositions on $MA(P)$ for the characterization theorem.

Proposition 2.3.3. *Let A, B, C be three maximal antichains of a partial order P such that $A \leq_{MA(P)} B \leq_{MA(P)} C$, then $A \cap C \subseteq B$.*

Proof. In the case where $A = B$ we have that $A \cap C \subseteq B$ and in the case $B = C$ we have that $A \cap C \subseteq B$. So we can assume that $A \neq B$, $B \neq C$ and as a consequence that $A <_{MA(P)} B <_{MA(P)} C$. Suppose for the sake of contradiction that $A \cap C \nsubseteq B$. So there exists $x \in (A \cap C)$-B. Since x does not belong to B there must exist a comparable element y in B to x. Using lemma 2.3.2 on A, B we establish that $x \leq_P y$. Using again lemma 2.3.2 on B, C we get that $y \leq_P x$. Since $x \leq y$ and $y \leq x$, by antisymmetry of P we deduce that $y = x$. Therefore x belongs to B which is a contradiction to the assumption that $x \in (A \cap C)$-B. □

Proposition 2.3.4. *Let A, B be two maximal antichains of a partial order P. Then $(A \cup B) - (A \wedge_{MA(P)} B) \subseteq A \vee_{MA(P)} B$.*

Proof. By lemma 2.2.3, we know that $A = (A \cap B) \cup S^-(A, B) \cup S^+(A, B)$ and $B = (A \cap B) \cup S^-(B, A) \cup S^+(B, A)$. Now using the definition of $A \wedge_{MA(P)} B$, we get that $(A \cap B) \cup S^-(A, B) \cup S^-(B, A) \subseteq (A \wedge_{MA(P)} B)$ and using the definition of $A \vee_{MA(P)} B$ we get that $(A \cap B) \cup S^+(A, B) \cup S^+(B, A) \subseteq (A \vee_{MA(P)} B)$. Thereby $(A \cup B) - (A \vee_{MA(P)} B) \subseteq A \wedge_{MA(P)} B$. □

2.4 Cliques structure of cocomparability graphs

In this section, we show that cocomparability graphs can be characterized by a lattice structure on its maximal cliques and then show that it extends the well-known characterization of the interval graphs by a linear ordering of its maximal cliques. Then, we state some corollaries on subclasses of cocomparability graphs.

Theorem 2.4.1. *$G = (V(G), E(G))$ is a cocomparability graph if and only if $C(G)$ the set of maximal cliques of G, can be equipped with a lattice structure L satisfying:*

(i) *For every A, B, C maximal cliques such that $A \leq_L B \leq_L C$, then $A \cap C \subseteq B$.*

(ii) *For every A, B maximal cliques, $A \vee B$ contains $(A \cup B)$-$(A \wedge B)$.*

Proof. For the forward direction, let P be a partial order on $V(G)$ which corresponds to a transitive orientation of $\overline{G}$. It must be noted that a maximal antichain of P forms a maximal clique of G. Let $L = MA(P)$. Proposition 2.3.3 shows that the first condition is satisfied. Proposition 2.3.4 shows that the second condition is satisfied.

For the reverse direction, we will prove the following claim, which shows that if there is a lattice structure on the maximal cliques of a graph G satisfying condition (i) and (ii), then we can transitively orient $\overline{G}$ and so G is a cocomparability graph.

Claim 2.4.2. *Let $G = (V(G), E(G))$ be a graph and $C(G)$ the set of maximal cliques of G such that $C(G)$ can be equipped with a lattice structure L satisfying conditions (i) and (ii). Then the binary relation $(V, \leq_P)$ defined by $x \leq_P y$ if and only if $xy \notin E(G)$ and there are maximal cliques C', C" of G with $C' \leq_L C$" and $x \in C'$, $y \in C$" is a partial order*

Proof. We will show that $\leq_P$ over $V(G)$ is a partial order. For that purpose, we start by showing that the relation is reflexive. After that if x, y are two vertices of G then we can find a couple of maximal cliques C', C" of G with $x \in C'$, $y \in C$" and ($C' \leq_L C$" or C" $\leq_L C'$). Then we will show the antisymmetry of $<_P$ and finally the transitivity.

Let x be a vertex of G. Because G is simple, we have that $xx \notin E(G)$. Let C_x be a maximal clique of G that contains x. We have that $C_x \leq_L C_x$ and so we deduce that $x \leq_P x$ which shows the reflexivity.

If we consider different vertices $x, y \in V(G)$ such that $xy \notin E(G)$, then x, y cannot be together in a maximal clique of G. Further, there exists at least two maximal cliques C_x, C_y such that $x \in C_x$ and $y \in C_y$. Assume $C_x \parallel_L C_y$. Since x, y cannot belong together to the supremum or the infimum of C_x and C_y, using condition (ii) the supremum necessarily contains x (resp. y) and the infimum will contain y (resp. x). Hence, we can derive $y \leq_P x$ (resp. $x \leq_P y$) using the pair of maximal cliques C_y, $C_x \wedge C_y$ (resp. the pair C_x, $C_x \wedge C_y$).

To show the antisymmetry of $\leq_P$, let us suppose for contradiction that $x \leq_P y$, $y \leq_P x$ and $x \neq y$. Then there exists C_x, C'_x, C_y, C'_y such that $x \in C_x$, $x \in C'_x$, $y \in C_y$, $y \in C'_y$, with $C_x \leq_L C_y$ and $C'_y \leq_L C'_x$. In the case where $C'_x \leq_L C_y$ then the three maximal cliques $C'_y \leq_L C'_x \leq_L C_y$ contradict condition (i) and if $C_y \leq_L C'_x$ then the three maximal cliques $C_x \leq_L C_y \leq_L C'_x$ contradict condition

(i) and so we deduce that $C'_x \parallel_L C_y$. But now using condition (ii) on C'_x, C_y, we deduce that in the supremum of C'_x and C_y we will find either x or y since they cannot belong together in a maximal clique. If it is y in $(C'_x \vee C_y)$, we have $C'_y \leq_L C'_x \leq_L (C'_x \vee C_y)$ that contradicts condition (i) and similarly if it is x in $(C'_x \vee C_y)$, then $C_x \leq_L C_y \leq_L (C'_x \vee C_y)$ contradicts condition (i). Thus if we have $x \leq_P y$ and $y \leq_P x$, we must have $x = y$.

Let us now examine the transitivity of $\leq_P$. Let x, y, z be three different vertices such that $x \leq_P y$ and $y \leq_P z$. Let us assume for contradiction that $xz \in E(G)$. Therefore there exists a maximum clique C_{xz} of G such that $x, z \in C_{xz}$. Let C_y be a maximal clique that contains y. But now because y is not linked to x it does not belong to C_{xz} and using condition (ii) on C_{xz} and C_y we have that either $y \in (C_{xz} \vee C_y)$ or $y \in (C_{xz} \wedge C_y)$. In the first case, by the definition of $\leq_P$ we have that $z \leq_P y$ and from the assumption, $y \leq_P z$. So using the antisymmetry of P on z, y we have that $z = y$, which contradicts our choice of z, y being different vertices. In the second case, by the definition of $\leq_P$, we have that $y \leq_P x$ and from the assumption, $x \leq_P y$. So using the antisymmetry on x, y, we have that $x = y$, which contradicts our choice of x, y being different vertices.

So assume that there exists three different vertices x, y, z such that $x \leq_P y$ and $y \leq_P z$. Now we show that $x \leq_P z$. We just have proved that xz must not be in $E(G)$. As $x \leq_P y$, there is a maximal clique C_x and a maximal clique C_y such that $x \in C_x$, $y \in C_y$ and $C_x \leq_L C_y$. Let C_z be a maximal clique such that $z \in C_z$. Using condition (ii) on C_z, C_y either $z \in (C_z \wedge C_y)$ or $z \in (C_z \vee C_y)$. In the case where $z \in (C_z \wedge C_y)$, using the definition of $\leq_P$ on z, y and the cliques C_y, $(C_z \wedge C_y)$ we get that $z \leq_P y$. Since $y \leq_P z$, using the antisymmetry of P we get $y = z$ thereby contradicting our assumption that y and z are different vertices. So z has to belong to $C_z \vee C_y$. Now we have $C_x \leq_L C_y \leq_L C_z \vee C_y$ and using the definition of $\leq_P$ on the vertices x, z and the cliques C_x, $C_z \vee C_y$, we deduce that $x \leq_P z$ which establishes the transitivity.

□

□

A natural question is : Is every lattice L satisfying the previous theorem a lattice $MA(P)$ for some partial order P that gives a transitive orientation of $\overline{G}$?

The answer is negative as can be seen in the example of Figure 2.4, however by adding a condition, we can establish when a lattice L is a lattice $MA(P)$.

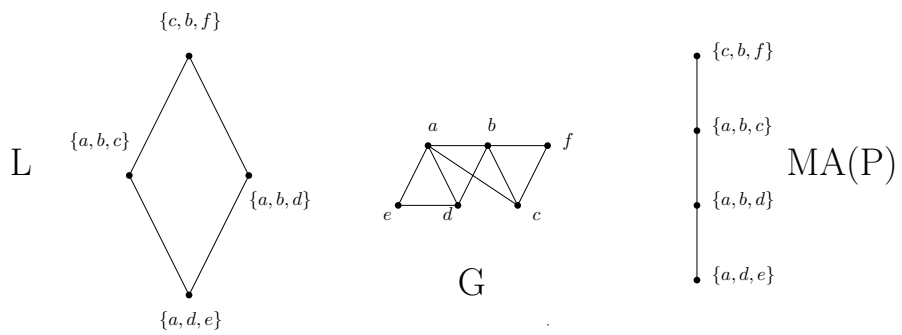


Figure 2.4: A graph G and a lattice L on $C(G)$ that satisfies condition (i) and (ii) of theorem 2.4.1. But L is not isomorphic to the lattice $MA(P)$ for a partial order P on $V(G)$ that corresponds to a transitive orientation of $\overline{G}$

Theorem 2.4.3. *If G is a cocomparability graph and L a lattice structure on $C(G)$ satisfying condition (i) and condition (ii) of theorem 2.4.1, then L is isomorphic to a lattice $MA(P)$ with P a transitive orientation of $\overline{G}$ if and only if the following condition is also satisfied:*

(iii) $\forall A, B \in C(G)$, $(A \cap B) \subseteq (A \vee_L B)$ and $(A \cap B) \subseteq (A \wedge_L B)$.

Proof. Suppose that L is isomorphic to a lattice $MA(P)$ with P a transitive orientation of $\overline{G}$. It is clear that (iii) is satisfied using the definition of $\wedge_{MA(P)}$ and $\vee_{MA(P)}$.

Conversely, using claim 2.4.2 we know that $P = (V(G), \leq_P)$ with the binary relation $\leq_P$ over $V(G)$ defined by $x \leq_P y$ if and only if $xy \notin E(G)$ and there are maximal cliques C', C" of G with $C' \leq_L C$" and $x \in C'$, $y \in C$" is a partial order. We now prove that L is isomorphic to the lattice $MA(P)$. So for this purpose we will show that for two maximal cliques A, B, $A \leq_L B$ if and only if $\forall x \in A$, $\exists y \in B$ with $x \leq_P y$ which is the definition of $MA(P)$. First we recall that since P is a transitive orientation of $\overline{G}$, any maximal clique of G corresponds to a maximal antichain in P. Because both L and P are partial orders, the relations are reflexive and the case where $A = B$ is clear.

Let A, B be two different maximal cliques of G such that $A \leq_L B$. Then $\forall x \in A - B$, x cannot be universal to $B - A$ because B is a maximal clique.

Therefore there exists $y \in B - A$ such that $xy \notin E(G)$ and so y is comparable with x in P. Furthermore we have that $x \neq y$ because $x \in A - B$ and $y \in B - A$. Using our definition of $\leq_P$ on x, y and the cliques A, B we get that $x \leq_P y$. For all $x \in A \cap B$ we also have that $x \leq_P x$ and so if $A \leq_L B$ then $\forall x \in A$, $\exists y \in B$ with $x \leq_P y$.

Let A, B be two different maximal cliques of G, such that $\forall x \in A$, $\exists y \in B$ with $x \leq_P y$. For the sake of contradiction assume that $A \parallel_L B$. Let us consider $A \vee_L B$. For a vertex $x \in A - B$, using the assumption we know that there exists $y \in B - A$ such that $x \leq_P y$ and so x must belong to $(A \wedge_L B)$ otherwise if $x \in (A \vee B)$, using the definition of $\leq_P$ on A, $(A \vee B)$ we have that $y \leq_P x$ and so $x = y$ which contradicts our choice of x and y. But now we have that $A - B \subseteq (A \wedge_L B)$ and using condition (iii) $(A \cap B) \subseteq (A \vee_L B)$ so $A \subseteq (A \wedge_L B)$. So either $A = (A \wedge_L B)$ and so $A \leq_L B$ which contradicts our choice of A and B, or $A \subsetneq (A \wedge_L B)$ which contradicts that A is a maximal antichain. □

So one can ask what is the relation between a lattice satisfying (i)and (ii) and a lattice satisfying (i),(ii) and (iii). From claim 2.4.2, for a given lattice structure L associated to a cocomparability graph and satisfying the conditions of theorem 2.4.1, we can define a partial order $P(V(G), \leq_P)$. From the previous proof, we know that if $A \leq_L B$ then $\forall x \in A$, $\exists y \in B$ with $x \leq_P y$ and so $A \leq_{MA(P)} B$. Therefore for a lattice L associated to a cocomparability graph G, there exists a transitive orientation P of $\overline{G}$ such that $MA(P)$ is an extension of L.

It must be noted that the last two theorems can be easily rewritten into a characterization of comparability by changing maximal clique into maximal independent set.

Let us now study the case of interval graphs.

Corollary 2.4.4. *[GH64] G is an interval graph if and only if $C(G)$ can be equipped with a total order T satisfying for every C_i, C_j, C_k maximal cliques such that $C_i \leq_T C_j \leq_T C_k$, then $C_i \cap C_k \subseteq C_j$.*

Proof. Using the last two theorems, we know that if G is a cocomparability graph then the set of maximal cliques of G can be equipped with a lattice structure satisfying conditions (i), (ii), (iii) and isomorphic to a lattice $MA(P)$ with P a transitive orientation of $\overline{G}$. Using property 2.2.5 which states that $MA(P)$ is a

chain if and only if P is an interval order, it can be noticed that condition (ii) of theorem 2.4.1 can never occur in a total order (chain). Therefore only condition (i) remains. □

We recall that we say that an ordering of cliques satisfies the consecutiveness property if for every vertex, the cliques that contains it occur consecutively.

Corollary 2.4.4 can be written equivalently as theorem 1.4.7:

The following theorem is a well known characterization of permutation graphs:

Theorem 2.4.5. *[DM41] G is a permutation graph if and only if G is a cocomparability and a comparability graph.*

Using the previous characterization of permutation graphs and the new characterization of cocomparability graphs, we obtain the following characterization of permutation graphs.

Corollary 2.4.6. *G is a permutation graph if and only if there exists a lattice structure satisfying (i) and (ii) on the set of maximal cliques and a lattice structure satisfying (i) and (ii) on the set of maximal independent sets.*

2.5 Conclusion and perspectives for further work

The main result of this section is the possibility to equip the set of maximal cliques of a cocomparability graphs with a lattice structure. This allow to define a huge variety of new subclasses of cocomparability graphs. For example we can define the class such that the lattice has a polynomial size or the lattice is distributive ...

This may have some interest for some problems like isomorphism. If we bound the size of the lattice or put some condition on the structure of the lattice, can we solve the isomorphism problem for cocomparability graphs in polynomial time?

Also for the recognition problem. Recognition of cocomparability graphs is an interesting problem and so far the best algorithm uses matrix multiplication. Can we define new subclasses of cocomparability graphs defined by properties of the lattice (for example $MA(P)$ is distributive) and such that we can solve the recognition problem faster? This problem will also be considered in the conclusion of chapter 4.

2. A NEW CHARACTERIZATION THEOREM FOR COCOMPARABILITY GRAPHS

3

Interval subgraphs of cocomparability graphs

3.1 Introduction

The study of interval subgraphs of cocomparability graphs was motivated by some results: Derek Corneil, Barnaby Dalton and Michel Habib showed that on a LDFS cocomp ordering the greedy algorithm to find a maximum path cover on an interval ordering works (see [CDH13]); Derek Corneil, Michel Habib and Ekkehard Kőhler showed that on a LDFS cocomp ordering the greedy algorithm to find a maximum independent set on an interval ordering works(see [CDHK]);Jérémie Dusart and Michel Habib showed that the multisweep LBFS$^+$ algorithm to find an interval ordering also find a cocomp ordering (see chapter 5). So a natural question arise: Do the cocomparability graphs have some kind of hidden interval structure? In this chapter, we establish some relationships between cocomparability graphs and their interval subgraphs.

In the previous chapter, we showed that we can equip the set of maximal cliques with a lattice structure. Let G be a cocomparability graph and σ a cocomp ordering of G. We recall that P_σ is the transitive orientation of $\overline{G}$ obtained using σ. Let us consider a maximal chain of $MA(P_\sigma)$, $\mathcal{C} = C_1 <_{MA(P_\sigma)} C_2 \cdots <_{MA(P_\sigma)} C_k$. The chain $C_1, C_2 \ldots C_k$ forms a chain of maximal cliques that respects Proposition 2.3.3 (consecutiveness condition). So using theorem 2.4.4, we deduce that this chain forms an interval subgraph of G. Therefore, we can see a cocomparability graph as a union of interval graphs. In this chapter, we develop an algorithm that computes a maximal chain of the lattice $MA(P_\sigma)$ and show

that it forms a maximal interval and chordal subgraphs. The problem of finding a maximal interval subgraph is the dual of the problem of finding a minimal interval completion (see [CT13, HSTV05]).

This algorithm also gives us a way to compute a minimal interval extension of a partial order. An interval extension of a partial order is an extension that is also an interval order. In [HMPR91, HMPR92], it has been proved that the maximal chains of $MA(P)$ are in a one-to-one correspondence with the minimal interval extensions. Therefore, our algorithm also allows us to compute a minimal interval extension of a partial order in $O(n^2)$.

This chapter is organized as follows. In section 3.2, we present a greedy algorithm, named Chainclique, to compute an interval subgraph using an ordering of the vertices. In section 3.3, we present a new graph search named LocalMNS. We will also prove that applying algorithm Chainclique on a LocalMNS cocomp ordering produces a maximal chain of $MA(P)$. In section 3.4, we show that a maximal chain of $MA(P)$ forms a maximal interval and chordal subgraphs. In section 3.5, we show that the problem of finding a maximum interval subgraph is NP-hard.

This work has been done in collaboration with Michel Habib.

3.2 Spanning interval graphs

This section is devoted to the presentation of algorithm 4, called Chainclique, which extracts an interval subgraph from an ordering. This algorithm is not new and has already been described in [COS09] for extracting the maximal cliques of an interval graph from an interval ordering. We generalize it to be able to be used on an arbitrary graph and ordering.

Definition 3.2.1. Let $G = (V(G), E(G))$ be a graph and σ an ordering of $V(G)$.

A graph $H = (V(G), E(H))$ with $E(H) \subseteq E(G)$ is a maximal interval subgraph for the ordering σ if and only if σ is an interval ordering for the graph H and $\forall S \subseteq E(G) - E(H)$, $S \neq \emptyset$, σ is not an interval ordering for the graph $H' = (V(G), E(H) \cup S)$.

Algorithm 4: Chainclique

Data: $G = (V(G), E(G))$ and an ordering σ
Result: a sequence of cliques C_1,...,C_j

```
j ← 0;
i ← 1;
C_0 ← ∅;
while i ≤ |V| do
    j ← j + 1 %{Starting a new clique}%;
    C_j ← {σ(i)} ∪ (N(σ(i)) ∩ C_{j−1});
    i ← i + 1;
    while i ≤ |V| and σ(i) is complete to C_j do
        C_j ← C_j ∪ {σ(i)} %{Augmenting the clique}%;
        i ← i + 1;
    end
end
Output C_1, ..., C_j;
```

As we will prove, Chainclique computes a maximal interval subgraph for an ordering. To this end, Chainclique computes a sequence of cliques that respects the consecutiveness condition. Chainclique tries to increase the current clique and when it cannot, it creates a new clique and sets it to be the new current clique. An other way to see it is that Chainclique discards all the edges $xz \in E(G)$ such that $\exists y$, $x <_\sigma y <_\sigma z$ and $xy \notin E(G)$.

It should be noticed that the cliques output by Chainclique are not necessarily maximal cliques of G, for example take a $P_3 = u,\ v,\ w$ and use Chainclique on the ordering $u,\ w,\ v$. It should also be noted that the algorithm works on an arbitrary graph and ordering. For an example, let us consider the graph in Figure 3.1 and the ordering $v_1,\ v_2,\ v_3,\ v_4,\ v_5,\ v_6,\ v_7$. Chainclique outputs $\{v_1,\ v_2\},\ \{v_2,\ v_3\},\ \{v_3,\ v_4\},\ \{v_4,\ v_5\},\ \{v_5,\ v_6,\ v_7\}$.

Now let us prove that Chainclique allows us to obtain a maximal interval subgraph for an ordering. We start with some terminology. For a chain $\mathcal{C} = C_1 <_{MA(P_\sigma)} C_2 \cdots <_{MA(P_\sigma)} C_k$, $G_\mathcal{C} = (V(G), E(\mathcal{C}))$ denotes the graph formed by the cliques $C_1, \ldots, C_k$ and for every vertex x we define $first[x]$ (resp. $last[x]$) as

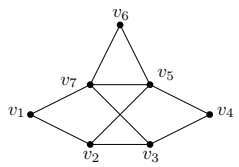


Figure 3.1: A graph G.

the first (resp. last) index of a clique of $\mathcal{C}$ that contains x. For a vertex x, $N_{\mathcal{C}}(x)$ is the neighborhood of x in the graph $G_{\mathcal{C}}$.

The proof is organized as follows. In the first proposition we prove that Chainclique outputs a sequence of cliques that respects the consecutiveness property. In the second proposition, we prove that the ordering given to Chainclique is an interval ordering for the sequence of cliques. In the last proposition we prove that the graph formed by the sequence is a maximal interval subgraph for the ordering.

Proposition 3.2.2. *For a graph G and an ordering σ, Chainclique outputs a sequence of cliques $\mathcal{C} = C_1, \ldots, C_k$ such that for every C_e, C_f, C_g, $1 \leq e \leq f \leq g \leq k$, $C_e \cap C_g \subseteq C_f$.*

Proof. We do the proof by induction on the cliques of $\mathcal{C}$ and the induction hypothesis is that at each step j if $x \in C_j - C_{j+1}$ then $x \notin C_{j'}$, $j' > j$

Since $C_0 = \emptyset$, the hypothesis is true for the initial case.

Assume that the hypothesis is true for the first $j \geq 1$ cliques. When we start to build the clique C_{j+1}, we add a vertex that has not been considered before and its neighborhood that belong to C_j. By doing so, we cannot add a vertex x to C_{j+1} such that $x \in C_i - C_j$ and $i < j$. When we increase the clique, we only add vertices that have not been considered before and so we cannot add a vertex x such that $x \in C_i - C_j$ and $i < j$, in C_{j+1}. Therefore the induction hypothesis is also verified at step $j + 1$. □

Therefore using characterization of interval graphs of corollary 2.4.4, Chainclique outputs a sequence of cliques that defines an interval subgraph.

Proposition 3.2.3. *For a graph G and an ordering σ, Chainclique outputs a sequence of cliques $\mathcal{C} = C_1, \ldots, C_k$ such that σ is an interval ordering for $G_{\mathcal{C}}$ and $\forall x \in C_i - C_j$, $\forall y \in C_j - C_i$, $i < j$, $x <_\sigma y$.*

Proof. Assume for contradiction that σ is not an interval ordering for $G_{\mathcal{C}}$. So there exists $u <_\sigma v <_\sigma w$ such that $uv \notin E(\mathcal{C})$ and $uw \in E(\mathcal{C})$. Let C_u be the first clique in which u appears, C_v be the first clique in which v appears and C_{uw} the first clique that contains both u and w. Because Chainclique considers the vertices in the order they appear in σ, the cliques C_u must appear in $\mathcal{C}$ before the clique C_v. Using the same argument the clique C_v must appear before the clique C_{uw}. But now C_u, C_v, C_{uw} contradict proposition 3.2.2.

Now assume for contradiction that $\exists x \in C_i - C_j$, $\exists y \in C_j - C_i$, $i < j$, $y <_\sigma x$. Now the vertices are considered by Chainclique in the order they appear in σ. Since $y <_\sigma x$, let C_g be the first clique in which y appears. We see that $g \leq i$. Since y belongs to C_g and C_j, using proposition 3.2.2 we know that $y \in C_i$. Therefore $y \notin C_j - C_i$, which contradicts our choice of y. Thus $\forall x \in C_i - C_j$, $\forall y \in C_j - C_i$, $i < j$, $x <_\sigma y$. □

We are ready to prove that the graph formed by the sequence is a maximal interval subgraph for the ordering.

Proposition 3.2.4. *For a graph G and an ordering σ, Chainclique outputs a sequence of cliques $\mathcal{C} = C_1, \ldots, C_k$ that induces a maximal interval subgraph for the ordering σ.*

Proof. Assume for contradiction that $\mathcal{C} = C_1, \ldots, C_k$ does not form a maximal interval subgraph for the ordering σ. Therefore there exists a non empty set of edges S such that σ is an interval ordering for the graph $H = (V(G), E(\mathcal{C}) \cup S)$. Let uv be an edge of S and assume without a loss of generality that $u <_\sigma v$. Let C_i be the last clique of $\mathcal{C}$ containing u and consider w the first vertex of C_{i+1}; clearly $uw \notin E$ and thus $w \neq v$. Now $u <_\sigma w <_\sigma v$ contradicts σ being an interval ordering for the graph H. □

Proposition 3.2.5. *Chainclique has complexity O(n+m).*

Proof. All the tests can be performed by enumerating once the neighborhood of a vertex. Since the sequence of cliques forms a subgraph, its size is bound by m. Therefore, Chainclique has complexity $O(n + m)$. □

3.3 Computing a maximal chain of MA(P)

In this section, we introduce a new graph search that will help us to compute a maximal chain of $MA(P)$ in a greedy fashion. This graph search will be baptized LocalMNS and when we use Chainclique on a LocalMNS cocomp ordering we get a maximal chain of MA(P).

First, we start by looking at the behavior of Chainclique on a LBFS and a LDFS ordering. Let us consider the graph in Figure 3.2. Applying the algorithm Chainclique on the LDFS ordering $1, 3, 2, 4, 6, 5$, we get the chain $\{1, 2, 3\}$, $\{1, 2, 4\}$ and $\{4, 5, 6\}$. We get the same result using the LBFS ordering $2, 3, 1, 4, 6, 5$. Thus, LBFS and LDFS does not help us to find a maximal chain of cliques of $MA(P)$ using Chainclique on them. So we now introduce LocalMNS (Algorithm 5).

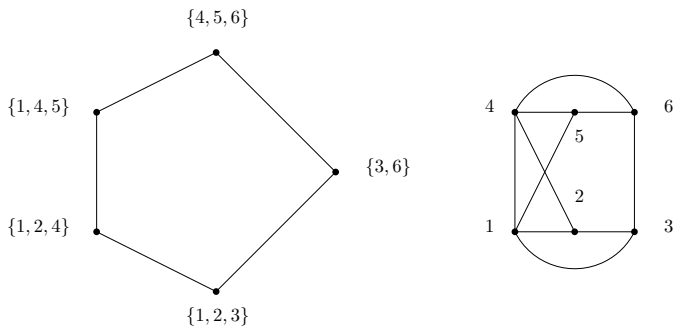


Figure 3.2: $MA(P)$ and the corresponding cocomparability graph G.

Algorithm 5: LocalMNS

Data: $G = (V, E)$
Result: a total ordering σ such that $\sigma(i)$ is the i'th visited vertex
$D_1 \leftarrow \emptyset$;
$V' \leftarrow V$ %{V' is the set of unchosen vertices}%;
$X \leftarrow \emptyset$;
for $i = 1$ *to* $|V|$ **do**
  v is chosen as a vertex with maximal neighborhood in D_i;
  $\sigma(i) \leftarrow v$;
  $V' \leftarrow V' - \{v\}$;
  $X \leftarrow X \cup \{v\}$;
  $D_{i+1} \leftarrow \{v\} \cup (N(v) \cap D_i)$;
end

This algorithm is very similar to the MNS algorithm. The only difference is in LocalMNS we are considering the neighborhood of the unvisited vertices only in D_i, which can be a strict subset of X (the unvisited vertices) and in the case of MNS we are considering the neighborhood in X. This is the reason for the name LocalMNS. Let us look at the behavior of $LocalMNS^+$ on example 3.2. Let $\tau = 5, 6, 4, 2, 3, 1$ be a cocomp ordering. $LocalMNS^+(G, \tau) = 1, 3, 2, 4, 5, 6$ is a cocomp ordering and Chainclique gives the maximal chain $\{1, 2, 3\}$, $\{1, 2, 4\}$, $\{1, 4, 5\}$ and $\{4, 5, 6\}$.

We now prove that Chainclique on a LocalMNS cocomp ordering outputs a maximal chain of cliques. Let G be a cocomparability graph and τ a cocomp ordering of G. The proof is organized as follows. We first show that $\sigma = LocalMNS^+(G, \tau)$ is a cocomp ordering. Then we describe the structure of a maximal chain of $MA(P_\sigma)$ and its relation to P_σ. Finally we prove that $Chainclique(G, \sigma)$ outputs a maximal chain of $MA(P_\sigma)$.

Lemma 3.3.1. *LocalMNS respects both the Graph Search Paradigm and the Private Neighbor Property.*

Proof. Using its definition it is clear that LocalMNS satisfies the GSP. When we

make our choice the elligible vertices is the set of vertices with maximal neighborhood in D_i and as with other maximal searches, it respects the PNP. □

Corollary 3.3.2. *If G is a cocomparability graph, τ is a cocomp ordering and $\sigma = LocalMNS^+(G, \tau)$ then σ is a cocomp ordering.*

Proof. Because of lemma 3.3.1, LocalMNS respects both the Graph Search Paradigm and the Private Neighbour Property and so using theorem 1.5.2, we deduce that σ is a cocomp ordering. □

Theorem 3.3.3. *Let G be a cocomparability graph and let σ be a cocomp ordering.*

$C_1 <_{MA(P_\sigma)} C_2 \cdots <_{MA(P_\sigma)} C_k$ is a maximal chain of $MA(P_\sigma)$ if and only if the following conditions are satisfied

- *C_1 is the set of sources of P_σ*
- *$1 < i \leq k$, C_i covers C_{i-1}*
- *C_k is the set of the sinks of P_σ*

Proof. For the forward direction, let $\mathcal{C} = C_1 <_{MA(P_\sigma)} C_2 \cdots <_{MA(P_\sigma)} C_k$ be a maximal chain of cliques of $MA(P_\sigma)$. Let C_S be the set of sources of P_σ. Since every source is incomparable with all the other sources, C_S is an antichain of P_σ. Every element that is not a source is comparable to at least one source and so C_S is a maximal antichain. We now show that for every maximal antichain A of P_σ, $C_S \leq_{MA(P_\sigma)} A$. Let A be a maximal antichain of P_σ; in the case $A = C_S$ then $C_S \leq_{MA(P_\sigma)} A$ and so we take $A \neq C_S$. For the sake of contradiction assume that $C_S \not\leq_{MA(P_\sigma)} A$. So we have two cases: either $A <_{MA(P_\sigma)} C_S$ or $A \parallel_{MA(P_\sigma)} C_S$. In the first case, let $y \in C_S - A$ and using lemma 2.3.1 on A and C_S we know there exists $x \in A - C_S$ such that $x <_{P_\sigma} y$. But now because $x <_{P_\sigma} y$ we contradict the fact that y is a source. In the second case, we know that there exists $(A \wedge_{MA(P)} C_S)$ such that $(A \wedge_{MA(P)} C_S) \leq_{MA(P_\sigma)} C_S$. Since $A \parallel_{MA(P_\sigma)} C_S$ we have $(A \wedge_{MA(P)} C_S) <_{MA(P_\sigma)} C_S$. But now we are back in the first case, which again gives us a contradiction. So for every maximal antichain A of P_σ, $C_S \leq_{MA(P_\sigma)} A$ and so we have that $C_S \leq_{MA(P_\sigma)} C_1$. Now if $C_S \neq C_1$ then we can add C_S at the beginning of the chain $C_1 <_{MA(P_\sigma)} C_2 \cdots <_{MA(P_\sigma)} C_k$, thereby contradicting the maximality of $\mathcal{C}$. Thus $C_S = C_1$.

Now assume for contradiction that for some $1 < i \leq k$, C_i does not cover C_{i-1}. Then there exists a maximal antichain B of P_σ such that $C_{i-1} <_{MA(P_\sigma)} B <_{MA(P_\sigma)} C_i$. But now the chain $C_1 \leq_{MA(P_\sigma)} \ldots C_{i-1} <_{MA(P_\sigma)} B <_{MA(P_\sigma)} C_i \cdots \leq_{MA(P_\sigma)} C_k$ is a chain such that $\mathcal{C}$ is a subchain which contradicts the maximality of $\mathcal{C}$.

Let C_P be the set of sinks of P_σ. Using the same argument as in the case of the set of sources, we can deduce that for every maximal antichain A of P_σ, $A \leq_{MA(P_\sigma)} C_P$. Now if $C_P \neq C_k$ then we can add C_P at the end of the chain $C_1 <_{MA(P_\sigma)} C_2 \cdots <_{MA(P_\sigma)} C_k$, thereby contradicting the maximality of $\mathcal{C}$. Thus $C_P = C_k$.

Conversely, assume for contradiction that $\mathcal{C}$ is not a maximal chain of cliques. Then we can add a maximal clique B to $\mathcal{C}$. There are three cases: B can be added at the beginning of $\mathcal{C}$; B can be added at the end of $\mathcal{C}$ or B can be added in the middle of $\mathcal{C}$. In the first case we have $B <_{MA(P_\sigma)} C_1$, which as shown previously contradicts C_1 being the set of sources. Similarly if $C_k <_{MA(P_\sigma)} B$ contradicts C_k being the set of sinks. In the last case, we have $C_{i-1} <_{MA(P_\sigma)} B <_{MA(P_\sigma)} C_i$ for some index i such that $1 < i \leq k$. But this is a contradiction to C_i covers C_{i-1}, which concludes the proof. □

Let us introduce some terminology to help us describe the behavior of Chainclique on an ordering σ. Let j_i be the first value of j such that $\sigma(i)$ belongs to C_j. Let $C_j^1, \ldots, C_j^{l_j}$ be the sequence to build the clique C_j. Let p_i be the first value of p such that $\sigma(i)$ belongs to the clique $C_{j_i}^p$.

We are ready to prove that Chainclique on a LocalMNS cocomp ordering σ outputs a maximal chain of $MA(P_\sigma)$. The proof is organized as follows. The next claim links Chainclique and LocalMNS. In proposition 3.3.5, we prove that the cliques output by Chainclique of a LocalMNS cocomp ordering are maximal cliques. In theorem 3.3.6, we prove that the chain is a maximal chain of $MA(P_\sigma)$.

Claim 3.3.4. *Let G be a cocomparability graph and let τ be a cocomp ordering.*

If $\sigma = LocalMNS^+(G, \tau)$ and $Chainclique(G, \sigma)$=$C_1, \ldots, C_k$ then for all values of i the set D_{i+1} computed by LocalMNS equals the set $C_{j_i}^{p_i}$ computed by Chainclique.

Proof. We do the proof by induction. The hypothesis is for all i, $D_{i+1} = C_{j_i}^{p_i}$.

Since $D_1 = \{\sigma(1)\}$ and $C_1^1 = \{\sigma(1)\}$, the hypothesis is true for $i = 1$.

Assume the hypothesis is true for the first i-1 vertices with $i > 1$. We have two cases: $\sigma(i)$ is complete to D_i or not. In the first case, LocalMNS increases the set D_i by adding $\sigma(i)$ and Chainclique will increase the clique $C_{j_{i-1}}^{p_{i-1}}$ by adding $\sigma(i)$. Therefore using the induction hypothesis at step i, we deduce that $D_{i+1} = C_{j_i}^{p_i}$. In the second case, LocalMNS sets $D_{i+1} = D_i \cap N(\sigma(i))$. In the same way, Chainclique creates a new clique $C_{j_i}^1 = C_{j_{i-1}}^{p_{i-1}} \cap N(\sigma(i))$. Therefore using the induction hypothesis at step i, we deduce that $D_{i+1} = C_{j_i}^{p_i}$.

□

Proposition 3.3.5. *Let G be a cocomparability graph and let τ be a cocomp ordering.*

If $\sigma = LocalMNS^+(G, \tau)$ then $Chainclique(G, \sigma) = C_1, \ldots, C_k$ is a chain of maximal cliques of $MA(P_\sigma)$ such that $C_1 <_{MA(P_\sigma)} C_2 <_{MA(P_\sigma)} \cdots <_{MA(P_\sigma)} C_k$.

Proof. We start by proving that $\mathcal{C} = C_1, \ldots, C_k$ are all maximal cliques of G and then $C_1 <_{MA(P_\sigma)} C_2 <_{MA(P_\sigma)} \cdots <_{MA(P_\sigma)} C_k$.

Assume for contradiction that some cliques are not maximal and let C_g be the first clique of the chain which is not a maximal clique of G. Let w be a vertex complete to C_g but $w \notin C_g$ and let w be the rightmost such vertex in σ. Let v be the first vertex in σ of $C_g - C_{g-1}$. We have two cases: either $v <_\sigma w$ or $w <_\sigma v$.

In the first case, let $u = \sigma(i)$ be first vertex of $C_{g+1} - C_g$. Since $w \notin C_g$, we must have $u <_\sigma w$. Using claim 3.3.4, we see that in LocalMNS at step i, $D_i = C_g$. But now at the time u has been chosen, w is complete to D_i but u is not. Thereby, contradicting the choice of LocalMNS to choose u.

In the second case,let C_h be the last clique in the chain with w. We must have that $h < g$ since $w <_\sigma v$. Since w is a neighbor of v, w is not in C_{g-1} otherwise $w \in C_g$ and we have that $h + 1 < g$. So now let consider the first vertex x of $C_{h+1} - C_h$ in the ordering σ. Since w does not belong to C_{h+1}, we know that $wx \notin E$ and since w is universal to C_g we have that $x \notin C_g$. Because w is the rightmost complete vertex to C_g, x must not be adjacent to some vertex y of C_g. Now either $x <_\sigma y$ or $y <_\sigma x$. In the first case we know that $w <_\sigma x <_\sigma y$ and w, x, y is an umbrella. Therefore σ is not a cocomp ordering, which is a contradiction to corollary 3.3.2. In the second case since y appears before x, we have that $first[y] \leq h + 1$. But now y belongs to $C_{first[y]}$ and C_g and so using property 3.2.2, we know that $y \in C_{h+1}$. This is a contradiction to $xy \notin E$. Thus

all the cliques in $\mathcal{C}$ are maximal cliques of G.

Let us now prove that $C_1 <_{MA(P_\sigma)} C_2 \cdots <_{MA(P_\sigma)} C_k$. For this purpose, we show that if $i < j$ then $\forall x \in C_i$, $\exists y \in C_j$ such that $xy \notin E$, $x \leq_\sigma y$. Assume first that $x \in C_i \cap C_j$ then we have that $xx \notin E$ and $x \leq_\sigma x$. Now assume that $x \in C_i - C_j$. Since C_j is a maximal clique of G and x does not belong to C_j, there exists $y \in C_j - C_i$ such that $xy \notin E$. We know by property 3.2.3 that $\forall x \in C_i - C_j$, $\forall y \in C_j - C_i$, $i < j$, $x <_\sigma y$. Therefore $x <_\sigma y$ and so $C_i <_{MA(P_\sigma)} C_j$. □

Let us prove that it forms a maximal chain now.

Theorem 3.3.6. *For a cocomparability graph G and a cocomp ordering τ of G, if $\sigma = LocalMNS^+(G, \tau)$ then $Chainclique(G, \sigma)$ is a maximal chain of maximal cliques of $MA(\sigma)$.*

Proof. Let $\mathcal{C} = C_1, \ldots, C_k$ be the chain of cliques output by $Chainclique(G, \sigma)$. In proposition 3.3.5, we proved that $\mathcal{C}$ forms a chain of maximal cliques of $MA(P_\sigma)$. Thereby we only have to prove now that the chain is maximal.

We will show in the next three claims that C_1 is the set of sources of P_σ, that C_j covers C_{j-1} and finally that C_k is the set of sinks of P_σ. Using theorem 3.3.3, we will be able to deduce that $C_1 <_{MA(P_\sigma)} C_2 <_{MA(P_\sigma)} \cdots <_{MA(P_\sigma)} C_k$ is a maximal chain of $MA(P_\sigma)$.

Claim 3.3.7. *C_1 is the set of sources of P_σ.*

Proof. For the initial case, let C_S be the set of sources of P_σ. We will show that $C_1 = C_S$ by proving that σ starts with all the vertices of C_S and only them. Since σ is a linear extension of P_σ, σ starts with at least one source. So we suppose without loss of generality that σ starts with a set of sources $S \subsetneq C_S$ and $S \neq \emptyset$. Now assume for contradiction that after S, we have a vertex x such that $x \notin C_S$. Let $i = \sigma^{-1}(x)$. All the sources after x are complete to S and so for LocalMNS to choose x, x must also be universal to S since at this step D_i is equal to S. Now since x does not belong to C_S, there is a vertex $v \in C_S$ that is comparable to x and because C_S is the set of sources of P_σ and since σ is a linear extension of P_σ, we must have that $v <_\sigma x$. And so v must belong to S. But now x cannot be complete to S, which is a contradiction. So σ starts with all the sources and only them. □

Claim 3.3.8. *C_j covers C_{j-1} for $1 < j \leq k$.*

Proof. Assume for contradiction that C_j does not cover C_{j-1} and this is the first time it happens in the chain. So C_g covers C_{g-1} for $1 < g \leq j-1$. Let A be a maximal clique of the lattice such that A covers C_{j-1} and $A <_{MA(P_\sigma)} C_j$. Using proposition 2.3.3 on C_{j-1}, C_j and A, we deduce that $C_{j-1} \cap C_j \subsetneq A$. We have two cases: either $A \subseteq C_j \cup C_{j-1}$ or $A \nsubseteq C_j \cup C_{j-1}$.

In the first case, since $C_{j-1} \cap C_j \subseteq A$, we have $C_{j-1} \cap C_j \subseteq A \cap C_{j-1}$. Assume for contradiction that $C_{j-1} \cap C_j = C_{j-1} \cap A$. Using $A \subseteq C_j \cup C_{j-1}$ and $C_{j-1} \cap C_j = C_{j-1} \cap A$ we can deduce that $A \subseteq C_j$, which contradicts the maximality of A. Therefore $C_{j-1} \cap C_j \subsetneq A \cap C_{j-1}$. Let $v = \sigma(i)$ be the leftmost vertex of $C_j - C_{j-1}$ in σ. We have again two cases: either $v \in A$ or $v \notin A$. In the first case, Chainclique set $C_j^1 = (N(v) \cap C_{j-1}) \cup \{v\}$. But since $C_{j-1} \cap C_j \subsetneq A \cap C_{j-1}$ and $v \in A$, we have $C_{j-1} \cap C_j \subsetneq N(v) \cap C_{j-1}$ and so $C_j^1 \nsubseteq C_j$. This is a contradiction to the behavior of Chainclique. In the second case, using claim 3.3.4 on v we deduce that the set D_i of localMNS equals C_{j-1}. But now let $x \in A - C_{j-1}$. Since $C_{j-1} \cap C_j \subsetneq A \cap C_{j-1}$, we have that $N(v) \cap D_i \subsetneq N(x) \cap D_i$. This is a contradiction to the choice of LocalMNS.

In the second case, let $x \in A - (C_j \cup C_{j-1})$. Assume for contradiction that there exists a maximal clique B such that $x \in B$ and $B <_{MA(P_\sigma} C_{j-1}$. Now $x \in B, A$ and $x \notin C_{j-1}$ contradicting proposition 2.3.3. Therefore x cannot appear in σ before the last vertex of C_{j-1}. Since $x \notin C_j$, $\exists y \in C_j$ such that $xy \notin E$. Using lemma 2.3.1 on x, y we deduce that $x <_{P_\sigma} y$ and since σ is a linear extension of P_σ we know that $x <_\sigma y$. But now since x appears after the last vertex of C_{j-1}, before the last of C_j and x is not complete to C_j, Chainclique must build a clique in the sequence between C_{j-1} and C_j, which is a contradiction.

□

Claim 3.3.9. *C_k is the set of sinks of P_σ.*

Proof. For the final case, we show that C_k is the set of sinks of P_σ. Assume for contradiction that x is a sink of P_σ and x does not belong to C_k. All the vertices belong to at least one clique of $\mathcal{C}$ and let $C_g = C_{last[x]}$. Since $x \notin C_k$, we have $g < k$. So x does not belong to C_{g+1}. But now let y be the first vertex in σ of $C_{g+1} - C_g$. Since $x \notin C_{g+1}$ we have $xy \notin E$ and $x <_\sigma y$. But this contradicts that x is a sink. □

□

So now we have a way to compute a maximal chain of $MA(P_\sigma)$. We now show that any maximal chain of $MA(P_\sigma)$ can be computed by LocalMNS.

Theorem 3.3.10. *For a cocomparability graph G and a transitive orientation P of $\overline{G}$, all the maximal chains of $MA(P)$ can be computed using Chainclique on some cocomp ordering produced by LocalMNS.*

Proof. Let $C_1 <_{MA(P)} ... <_{MA(P)} C_k$ be a maximal chain of $MA(P)$.

Let us take the ordering τ as the interval ordering for the maximal chain of cliques. Now using $LocalMNS^+$ on τ^{-1}, we get τ. So τ is a LocalMNS cocomp ordering and using Chainclique on τ we get $C_1 <_{MA(P)} ... <_{MA(P)} C_k$. □

In the next section, we will show that a restriction of LocalMNS, namely LocalMCS, has complexity $O(n^2)$.

3.4 Maximal chordal and interval subgraphs

A graph $H = (V(H), E(H))$ with $E(H) \subseteq E(G)$ is a maximal chordal subgraph if and only if H is a chordal graph and $\forall S \subseteq E(G) - E(H)$, $S \neq \emptyset$, $H' = (V(H), E(H) \cup S)$ is not a chordal graph. In the same way, a graph $H = (V(H), E(H))$ with $E(H) \subseteq E(G)$ is a maximal interval subgraph if and only if H is an interval graph and $\forall S \subseteq E(G) - E(H)$, $S \neq \emptyset$, $H' = (V(H), E(H) \cup S)$ is not an interval graph.

The problem of finding a maximal chordal subgraph has been looked at in [DSW88] and an algorithm with complexity $O(nm)$ has been proved. In this section, we show that a maximal chain of $MA(P)$ forms a maximal interval subgraph but also a maximal chordal subgraph. At the end of the section we will show that a restriction of LocalMNS, namely LocalMCS, has complexity $O(n^2)$. Since Chainclique has complexity $O(n + m)$, we will have a way to compute a maximal interval and chordal subgraph of a cocomparability graph in $O(n^2)$.

It must be noted that not all maximal chordal subgraphs of a cocomparability graph are an interval graph as shown in Figure 3.3.

We now show that a maximal chain of $MA(P_\sigma)$ is a maximal interval subgraph and a maximal chordal subgraph.

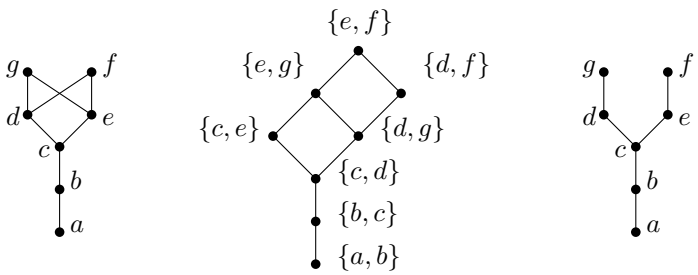


Figure 3.3: From left to right: a cocomparability graph, along with one of its lattice and a maximal chordal subgraph that is not an interval graph.

Theorem 3.4.1. *Let G be a cocomparability graph and let σ be a cocomp ordering.*

If $\mathcal{C} = C_1 <_{MA(P_\sigma)} C_2 \cdots <_{MA(P_\sigma)} C_k$ is a maximal chain of $MA(P_\sigma)$ then $G_{\mathcal{C}}$ is a maximal interval subgraph and a maximal chordal subgraph of G .

Proof. Assume for contradiction that $G_{\mathcal{C}}$ is not a maximal chordal subgraph. Let S be a set of edges such that the graph $H = (V(G), E(\mathcal{C}) \cup S)$ is a maximal chordal subgraph. In the proof, we will carefully choose an edge uv and show that we can find an induced path from u to v of length at least 3 in H. Therefore it will prove that $G_{\mathcal{C}}$ is a maximal chordal subgraph. Since interval graphs is a subclass of chordal graphs and $G_{\mathcal{C}}$ is an interval graph, we will deduce that $G_{\mathcal{C}}$ is also a maximal interval subgraph.

We start by proving two claims.

Claim 3.4.2. *Let G be a cocomparability graph, let σ be a cocomp ordering and let u, v be two vertices of G such that $uv \in E$.*

If C_u, C_v are maximal cliques of G such that $u \in C_u$, $v \notin C_u$, $v \in C_v$, $u \notin C_v$, $C_u <_{MA(P_\sigma)} C_v$ then there exists a maximal clique C_{uv} such that $u, v \in C_{uv}$ and $C_u <_{MA(P_\sigma)} C_{uv} <_{MA(P_\sigma)} C_v$

Proof. Since $uv \in E$ there must exist a maximal clique C^0_{uv} of G that contains u and v. We now define $C^1_{uv} = C^0_{uv} \vee_{MA(P_\sigma)} C_u$ and show that u, v belong to C^1_{uv}. We have three cases: $C_u <_{MA(P_\sigma)} C^0_{uv}$; $C^0_{uv} <_{MA(P_\sigma)} C_u$; $C_u \parallel_{MA(P_\sigma)} C^0_{uv}$. In the first case, we see that $C^1_{uv} = C^0_{uv}$ and so u, v belong to C^1_{uv}. In the second case, v belongs to C^0_{uv} and C_v but not to C_u and so $C^0_{uv} <_{MA(P_\sigma)} C_u <_{MA(P_\sigma)} C_v$ contradicts proposition 2.3.3 (consecutiveness condition). Therefore this case

cannot happen. In the last case, using the definition of $\vee_{MA(P_\sigma)}$ on C^0_{uv} and C_u, we see that u must belong to C^1_{uv} because it belongs to $C^0_{uv} \cap C_u$. Using again the definition of $\vee_{MA(P_\sigma)}$ on C^0_{uv} and C_u, we see that v must belong to C^1_{uv} otherwise v would have belong to $C^0_{uv} \wedge C_u$ and $(C^0_{uv} \wedge C_u) <_{MA(P_\sigma)} C_u <_{MA(P_\sigma)} C_v$ would contradict proposition 2.3.3 (consecutiveness condition).

We finish the proof of the claim by showing that $C_{uv} = C^1_{uv} \wedge C_v$ satisfies u, $v \in C_{uv}$ and $C_u <_{MA(P_\sigma)} C_{uv} <_{MA(P_\sigma)} C_v$. From the choice of C_{uv} we know $C_{uv} <_{MA(P_\sigma)} C_v$. We have three cases: $C^1_{uv} <_{MA(P_\sigma)} C_v$; $C_v <_{MA(P_\sigma)} C^1_{uv}$; $C^1_{uv} \parallel_{MA(P_\sigma)} C_v$. In the first case, we see that $C_{uv} = C^1_{uv}$ and so $u, v \in C_{uv}$ and $C_u <_{MA(P_\sigma)} C_{uv}$. In the second case, u belongs to C^1_{uv} and C_u but not to C_v and so $C_u <_{MA(P_\sigma)} C_v <_{MA(P_\sigma)} C^1_{uv}$ contradicts proposition 2.3.3 (consecutiveness condition). Therefore this case cannot happen. In the last case, using the definition of $\wedge_{MA(P_\sigma)}$ on C^1_{uv} and C_v we see that v must belong to C_{uv} since it belongs to $C^1_{uv} \cap C_v$. Using theorem 2.4.1 on C^1_{uv} and C_u, we get that u must belong to C_{uv} otherwise u would have belong to $(C^1_{uv} \vee C_v)$ and $C_u <_{MA(P_\sigma)} C_v <_{MA(P_\sigma)} (C^1_{uv} \vee C_v)$ would contradict proposition 2.3.3 (consecutiveness condition). Since C_{uv} is defined as $C^1_{uv} \wedge C_v$ by the definition of the lattice $C_u <_{MA(P_\sigma)} C_{uv}$. □

Claim 3.4.3. *Let G be a cocomparability graph, σ be a cocomp ordering, $\mathcal{C} = C_1 <_{MA(P_\sigma)} C_2 \cdots <_{MA(P_\sigma)} C_k$ be a maximal chain of $MA(P_\sigma)$, u, v be two vertices of G such that $uv \in E$.*

If $last[u] < first[v]$ then $\exists x$, y such that $first[x] \leq last[u] < first[y] \leq last[x] < first[v]$, $last[x] \leq last[y]$ and $yv \in E$.

Proof. Let $C_u = C_{last[u]}$ and $C_v = C_{first[v]}$. Since $u \in N(v) - N_{\mathcal{C}}(v)$, we deduce that $uv \notin E(\mathcal{C})$. Furthermore, since $u <_\sigma v$, C_u, $C_v \in \mathcal{C}$, we see that $C_u <_{MA(P_\sigma)} C_v$. Using the previous claim on C_u, C_v, we deduce there exists C_{uv} a maximal clique of G such that $C_u <_{MA(P_\sigma)} C_{uv} <_{MA(P_\sigma)} C_v$. Since $C_1, \ldots, C_k$ is a maximal chain of cliques and $uv \notin E(\mathcal{C})$ we further know that $C_{uv} \notin \mathcal{C}$. Since $\mathcal{C}$ is a maximal chain, $C_u <_{MA(P_\sigma)} C_{uv} <_{MA(P_\sigma)} C_v$ and $C_{uv} \notin \mathcal{C}$, we know that there exists a maximal clique D_1 in $\mathcal{C}$ such that $C_u <_{MA(P_\sigma)} D_1 <_{MA(P_\sigma)} C_v$ and D_1 covers C_u otherwise we contradict the maximality of the chain. We have three cases: $C_{uv} <_{MA(P_\sigma)} D_1$; $D_1 <_{MA(P_\sigma)} C_{uv}$; $D_1 \parallel_{MA(P_\sigma)} C_{uv}$. In the first cases, this contradicts the assumption that D_1 covers C_u. So this case cannot

happen. In the second case, we have $u \in C_u, C_{uv}$ and $u \notin D_1$ which contradicts proposition 2.3.3 (consecutiveness condition) and so this case cannot happen. So we are left with the last case. Since D_1 covers C_u and $D_1 \parallel_{MA(P_\sigma)} C_{uv}$ we now show that $D_1 \wedge C_{uv} = C_u$. Assume it is not the case then we will have $C_u <_{MA(P_\sigma)} (D_1 \wedge C_{uv}) <_{MA(P_\sigma)} D_1$ which contradicts that D_1 covers C_u. So we are in the situation described in Figure 3.4.

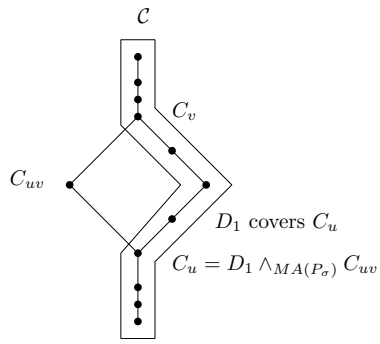


Figure 3.4

Since we chose C_v as the first maximal clique in $\mathcal{C}$ that contains v, v is not universal to D_1 and let x be a vertex of maximum last among $D_1 - N(v)$. So $last[x] < first[v]$ Let us show that x belongs to C_u. Assume that x belongs to $C_{uv} \vee D_1$, then $v \notin C_{uv} \vee D_1$ and using proposition 2.3.4 we deduce that $v \in C_{uv} \wedge D_1$. But now v belongs to C_u which contradicts that uv does not belong to $G_{\mathcal{C}}$. Thus x is a vertex such that $first[x] \leq last[u] < last[x] < first[v]$.

Let $C_x = C_{last[x]}$. Since $\mathcal{C}$ is an interval graph and $v \notin N(x)$, we see $C_x <_{MA(P_\sigma)} C_v$. Using the same argument as in the case of D_1, we also see that $C_x \parallel_{MA(P_\sigma)} C_{uv}$. From the lattice definition we have that $(C_x \vee_{MA(P_\sigma)} C_{uv}) \leq_{MA(P_\sigma)} C_v$ and $C_u \leq_{MA(P_\sigma)} (C_x \wedge C_{uv})$. Using proposition 2.3.3 (consecutiveness condition) on $C_{uv} \leq_{MA(P_\sigma)} (C_x \vee C_{uv}) \leq_{MA(P_\sigma)} C_v$ we see that $v \in (C_x \vee C_{uv})$. Using proposition 2.3.3 (consecutiveness condition) on $C_u \leq_{MA(P_\sigma)} (C_x \wedge C_{uv}) \leq_{MA(P_\sigma)} C_{uv}$ we see that $u \in (C_x \wedge C_{uv})$. Since $u \notin C_x$, u is not universal to C_x and let y be a vertex of maximum last among $C_x - N(u)$. So $last[u] < first[y] \leq last[x]$ and $last[x] \leq last[y]$. Since $u \in (C_x \wedge C_{uv})$ and $y \notin N(u)$, using proposition 2.3.4 we deduce that $y \in C_x \wedge C_{uv}$ and so $y \in N(v)$. So we have $yv \in E$. $\square$

We now carefully choose an edge $uv \in S$ and show that we can find an induced path of length at least 3 in H from u to v. Let uv be an edge of S such that $last[u] < first[v]$ and $\nexists xy \in S$, $last[x] < first[y]$ and $((last[u] < last[x]$ and $first[y] \leq first[v])$ or $(last[u] \leq last[x]$ and $first[y] < first[v]))$.

Using the previous claim on u and v, we know that there exists x_1 and y_1 such that $first[x_1] \leq last[u] < first[y_1] \leq last[x_1] < first[v]$, $last[x_1] \leq last[y_1]$ and $y_1v \in E$. We choose x_1 and y_1, the vertices of maximum last among the ones satisfying the conditions. By our choice of uv we know that $x_1v \notin E(H)$ and $uy_1 \notin E(H)$. Now we have two cases: either $first[v] \leq last[y_1]$ or $last[y_1] < first[v]$. In the first case, we have that u, x_1, y_1, v is an induced path of length 3 in H, and so an induced C_4, which contradicts H to be a chordal graph. In the second case, we apply the previous claim on y_1, v and deduce that there exists x_2 and y_2 such that $first[x_2] \leq last[y_1] < first[y_2] \leq last[x_2] < first[v]$, $last[x_2] \leq last[y_2]$ and $y_2v \in E$. We choose x_2 and y_2, the vertices of maximum last among the ones satisfying the conditions. By our choice of uv we know that $x_2v \notin E(H)$, $uy_2 \notin E(H)$ and $y_1, y_2 \notin E(H)$. Since we choose x_1 to be the vertex of maximum last we know that $x_2u \notin E(H)$. Since we choose y_1 to be a vertex of maximum last we know that $x_1x_2 \notin E$. Now we have again two cases: either $first \leq last[y_1]$ or $last[y_1] < first[v]$. In the first case, u, x_1, y_1, x_2, y_2, v is an induced path of length 5 in H, and so a induced C_6, which contradicts H to be chordal. In the second case, we do the same argument again. By keeping doing it, we always find an induced path from u to v of length at least 3 in H. Therefore H cannot be chordal.

□

Theorem 3.4.4. *A maximal interval subgraph and a maximal chordal subgraph of a cocomparability graph can be computed with complexity $O(n^2)$.*

Proof. The algorithm consists of finding a cocomp ordering, then performing a $LocalMCS^+$ and then using Chainclique. A cocomp ordering can be found in $O(n+m)$ [MS99] and Chainclique has complexity $O(n+m)$. So the bottleneck of this algorithm lies in $LocalMCS^+$.

We now describe an implementation of $LocalMCS^+$ with complexity $O(n^2)$. LocalMCS works the same ways as LocalMNS except that at each step i, instead of choosing a vertex of maximal neighborhood in D_i, we choose a vertex with max-

imum neighborhood in D_i. Let τ be a cocomp ordering and $\sigma = LocalMCS(\tau, G)$.

For each vertex x of V', we define $card[x]$ as the cardinality of the intersection of $N(x)$ with the current set D_i. We initialize it at 0. As soon as we add a vertex v to D_i, for each vertex $y \in V' \cap N(v)$, $card[y] = card[y] + 1$. As soon as we remove a vertex u of D_i, for each vertex $y \in V' \cap N(u)$, $card[y] = card[y] - 1$. Since as soon as a vertex is removed from D_i it cannot appear again, during the execution of $LocalMCS^+$, we enumerate the neighborhood of each vertex at most 2 times. Therefore the complexity to update the set D_i is $O(m + n)$.

To find the maximum one, we enumerate following τ all the vertices to find the rightmost unvisited vertices with maximum neighborhood. Since we have to do it n times, the complexity to find the maximum is $O(n^2)$.

□

We conjecture that LocalMCS can be implemented in $O(n + m)$.

3.5 Maximum interval subgraph

After looking at the problem of finding a maximal interval subgraph, it is natural to look at the problem of finding the maximum interval subgraph. For a graph $G = (V(G), E(G))$ we want to find a minimum set of edges whose deletion makes the graph an interval graph. In this section, we will show that this problem is NP-hard when G is a cocomparability graph.

In order to show that, we will just link this problem to the problem of the minimal fill in. The proof will in fact, widely rely on the proof of the NP-compleness of the minimum fill-in problem presented in the paper [Yan81]. We will first introduce some class of graphs.

A graph $G = (V(G), E(G))$ is a bipartite graph if and only if $V(G)$ can be partitioned into two independent set $S_1(G)$ and $S_2(G)$. A bipartite graph will be also denoted by $G = (S_1(G), S_2(G), E(G))$. A bipartite graph $G = (S_1(G), S_2(G), E(G))$ is a chain graph if and only if there exists an ordering π of $S_1(G)$ such that for $i,\ j$, $1 \leq i < j \leq |S_1$, $N(\pi(j)) \subseteq N(\pi(i))$.

Lemma 3.5.1. *[Yan81] A bipartite graph is a chain graph if and only if it does not contain a pair of independent edges.*

Theorem 3.5.2. *[Yan81] It is NP-complete for a bipartite graph $G = (S_1(G), S_2(G), E(G))$ to find the minimum number of edges whose addition gives a chain graph.*

The previous theorem can be rewritten this way:

Theorem 3.5.3. *It is NP-complete for a cobipartite graph $G = (S_1(G), S_2(G), E(G))$ to find the minimum number of edges whose removal gives an interval graph.*

Proof. This theorem is just a reformulation of the previous theorem using lemma 3.5.1 and passing to the complement. Using lemma 3.5.1, a chain graph can be see as an interval partial order. Therefore the addition to a bipartite graph $G = (S_1(G), S_2(G), E(G))$ to give a chain graph is equivalent to the removal to a cobipartite graph $G = (S_1(G), S_2(G), E(G))$ to give an interval graph. □

Since cobipartite graphs are cocomparability graphs, we deduce that:

Corollary 3.5.4. *It is NP-hard for a cocomparability graph to find the minimum number of edges whose removal gives an interval graph.*

3.6 Conclusions and perspectives for further work

In this chapter, we have exhibited some relationships shared by cocomparability graphs and interval graphs. But, we still have not answered the questions why some greedy algorithms for interval graph work on some special cocomp ordering and why some others do not. Does there exist some greedoid structure for some problems in cocomparability graph that explains why these greedy algorithms work? So far we still have no good answer for this question.

3. INTERVAL SUBGRAPHS OF COCOMPARABILITY GRAPHS

4

Simplicial vertices and clique separators in cocomparability graphs

4.1 Introduction

A vertex $v \in G$ is a simplicial vertex if and only if the graph induced by $N(v)$ is a clique. Simplicial vertices play an important role in gaussian elimination, see [Gol04, RT75]. We present an algorithm to find them all in linear time in the case that the input graph is a cocomparability graph.

For a connected graph G, $S \subseteq V(G)$ is a **separator** if $G - S$ is disconnected. For two vertices $a,\ b \in V(G)$, S is an ab-separator if a and b are in different connected components of $G - S$. S is a minimal ab-separator if no proper subset of S is an ab-separator. S is a **minimal clique separator** of a graph G if S is a clique and there is some vertex a, b such that S is a minimal ab-separator. An atom is a maximal induced subgraph that admits no clique separator.

The clique separators are interesting for a graph G since it allows the use of the Divide-and-Conquer approach for some NP-complete problem such that graph coloring or maximum clique, see [Tar85, BPS10]. So far the approach used to compute the clique separators of a graph G always used the minimal triangulation as a first step, i.e. a completion of the graph into a chordal graph. This approach has the inconvenience of making the graph grow and does not allow solving the problem in linear time. The algorithm we present solves the problem for cocomparability graphs, without using the minimal triangulation and runs in

linear time. So far it is the only algorithm known for this problem that runs in linear time for a class different from the chordal graphs class.

This chapter is organized as follows. Section 4.2 is devoted to proving an important theorem for both the clique separator algorithm and the simplicial vertices algorithm. Section 4.3 is devoted to solving the simplicial vertices problem. Section 4.4 is devoted to solving the problem of finding the clique separators of G.

This work has been done in collaboration with Michel Habib.

4.2 Fully comparable clique and MNS ordering

In a lattice $L = (X, \leq_L)$, an element $e \in X$ is said fully comparable if and only if for every $u \in X$, $e \leq_L u$ or $u \leq_L e$.

We now prove that if σ is an MNS cocomp ordering of G then all the fully comparable maximal cliques of $MA(P_\sigma)$ belong to the sequence of cliques obtained using Chainclique(G,σ). As we will show, these cliques play a decisive role in the problem of finding the simplicial vertices and the clique separators of G.

Theorem 4.2.1. *Let $G = (V(G), E(G))$ be a cocomparability graph and σ be a MNS cocomp ordering of the vertices of G.*

If C_b is a maximal clique such that C_b is fully comparable in $MA(P_\sigma)$ then C_b is a maximal clique of the chain output by $Chainclique(G, \sigma)$.

Proof. Let C_b be a fully comparable maximal clique in $MA(P_\sigma)$. We define V_{C_b} to be $\{x \in V | \exists C_x$ such that $x \in C_x$ and $C_x \leq_{MA(P_\sigma)} C_b\}$.

Let us first prove that the ordering σ starts with all the vertices of V_{C_b}. For the sake of a contradiction, let's assume that we have in σ a vertex v such that $\exists C_v$, $v \in C_v$ and $C_v >_{MA(P_\sigma)} C_b$ before a vertex $x \in V_{C_b}$ and v is the leftmost such vertex. Let C_x be a maximal clique such that $x \in C_x$ and $C_x \leq_{MA(P_\sigma)} C_b$. We have two cases, either $xv \notin E$ or $xv \in E$. Assume we are in the first case. Using lemma 2.3.1 on x, v, we deduce that $x <_{P_\sigma} v$ and since σ is a linear extension of P_σ, we know that $x <_\sigma v$ contradicting our choice of v. Assume we are in the second case. We prove that $N(v) \cap V_{C_b} \subseteq N(x) \cap V_{C_b}$. Since $xv \in E$ there exists a maximal clique D such that $\{x, v\} \subseteq D$ and since $v \notin V_{C_b}$ we know

$C_b <_{MA(P_\sigma)} D$. Using proposition 2.3.3 on C_x, C_b, D we deduce that $x \in C_b$. Let u be a vertex of $N(v) \cap V_{C_b}$. Since $uv \in E$ there exists a maximal clique A such that $\{u, v\} \subseteq A$ and since $v \notin V_{C_b}$ we know $C_b <_{MA(P_\sigma)} A$. Let C_u be a maximal clique such that $u \in C_u$ and $C_u \leq_{MA(P_\sigma)} C_b$. Using proposition 2.3.3 on C_u, C_b, A we deduce that $u \in C_b$. Therefore $ux \in E$ and so $N(v) \cap V_{C_b} \subseteq N(x) \cap V_{C_b}$. But now at the time when v was chosen, the label of v can be at most equal to the label of x since it's the first time. Now C_b is a maximal clique and since $v \notin C_b$, there must exist a vertex $w \in C_b$ such that $wv \notin E$. From a previous fact we know $w <_\sigma v$. Since $x \in C_b$, $wx \in E$ and so the label of x is strictly greater than the label of v when v was chosen which is a contradiction to the choice of MNS. Thus σ starts with all the vertices of V_{C_b}.

Since σ starts with all the vertices of V_{C_b}, the ordering of the vertices of V_{C_b} induced by σ is a MNS cocomp ordering for the graph induced by V_{C_b}. Let P_{C_b} be the transitive orientation of the complement of the graph induced by V_{C_b} obtained using σ. To prove that C_b belongs to the interval graph computed by *Chainclique* we will prove that the last clique that Chainclique computes using the ordering induced by the vertices of V_{C_b} is the set of sinks of P_{C_b}, which is equal to C_b. Let $C_1, \ldots, C_k$ be the chain of cliques that Chainclique computes using the ordering induced by the vertices of V_{C_b}. Assume for contradiction that x is a sink and $x \notin C_k$. Let C_g be the last clique that contains x. Since $x \notin C_k$ we have that $g < k$. Let y be the first vertex in σ that belongs to $C_{g+1} - C_g$. Since $x \notin C_{g+1}$, we must have that $xy \notin E$ and so $x <_{P_{C_b}} y$, therefore contradicting the assumption that x is a sink. □

The theorem is not true if we do not have the condition that the ordering σ is a MNS cocomp ordering. Take a P_3, u, v, w and the ordering $u < w < v$ (see Figure 4.1). The algorithm cannot output $\{u, v\}$ which satisfies the property.

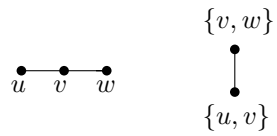


Figure 4.1: A P_3 and its lattice.

4.3 Simplicial vertices in cocomparability graphs

In this section we present an algorithm to find all the simplicial vertices in a cocomparability graph. The correctness of the algorithm will mainly rely on theorem 4.3.1 and theorem 4.2.1.

Theorem 4.3.1. *Let G be a cocomparability graph and σ a cocomp ordering.*

If v is a simplicial vertex then there exists a maximal clique C_v such that $v \in C_v$ and C_v is fully comparable in $MA(P_\sigma)$.

Proof. Clearly a simplicial vertex belongs to a unique maximal clique. For a simplicial vertex v let us denote by C_v the maximal clique such that $v \in C_v$. Assume that there exists a maximal clique D such that $D \parallel_{MA(P_\sigma)} C_v$. Since $MA(P_\sigma)$ is a lattice, there exists $D \vee C_v$ and $D \wedge C_v$ two other maximal cliques. Using the definition of $D \vee C_v$ and $D \wedge C_v$, v belongs to either $D \vee C_v$ or $D \wedge C_v$. Therefore C_v is not the only maximal clique that contains v, contradicting v is a simplicial vertex. □

To compute the simplicial vertices we need for each vertex a couple of values. We compute the value $first$ and $last$ define by $first[v] = min\{i|v \in C_i\}$ and $last[v] = max\{i|v \in C_i\}$. The value forward will be the last clique in which the vertex has a neighbour in the graph G but not in the interval graph and so be define by $forward[v] = max\{\{first[u]|u \in N(v)$ and $first[u] > last[v]\} \cup \{last[v]\}\}$. It must be noted that $forward[v] \geq last[v]$. The value backward will be the first clique in which the vertex has a neighbour in the graph G but not in the interval graph and so we define $backward[v]$ to be $min\{\{last[u]|u \in N(v)$ and $last[u] < first[v]\} \cup \{first[v]\}\}$. It must be noted that $backward[v] \leq first[v]$.

Theorem 4.3.2. *Let G be a cocomparability graph and σ be an MNS cocomp ordering of the vertices of G.*

Algorithm 6 outputs all the simplicial vertices of G.

Proof. Using theorem 4.2.1 we deduce that all the simplicial vertices and their whole neighbourhood belong to one clique of the sequence output by Chainclique.

Algorithm 6: Simplicial vertices in a cocomparability graph

Data: $G = (V(G), E(G))$ and an MNS cocomp ordering σ of $V(G)$
Result: The simplicial vertices of G
Compute the sequence $C_1, ..., C_k$ using $Chainclique(G, \sigma)$;
Compute first, last, backward, forward for every vertex of $V(G)$;
$S \leftarrow \emptyset$ %{The simplicial vertices}%;
for $(i \leftarrow 1$ **to** $n)$ **do**
  if $forward[\sigma(i)] == backward[\sigma(i)]$ **then** $S = S \cup \{\sigma(i)\}$;
end
Output S;

So to test if a vertex is a simplicial vertex, we have to check that a vertex belongs to only one clique in the sequence and has no neighbours in the rest of the interval graph. Because of our definition, for any vertex v we have that $last[v] \leq forward[v]$ and $backward[v] \leq first[v]$. So for any simplicial vertex $\sigma(i)$ we have that $forward[\sigma(i)] == backward[\sigma(i)] == last[\sigma(i)] == first[\sigma(i)]$. If a vertex $\sigma(i)$ is not simplicial then either it belongs to more than one clique in the sequence and so $last[\sigma(i)] \neq first[\sigma(i)]$ implying $forward[\sigma(i)] \neq backward[\sigma(i)]$ or it has a neighbor outside the sequence and so $last[\sigma(i)] < forward[\sigma(i)]$ or $backward[\sigma(i)] < first[\sigma(i)]$ implying $forward[\sigma(i)] \neq backward[\sigma(i)]$. Therefore our algorithm successfully finds all the simplicial vertices of a cocomparability graph. □

Theorem 4.3.3. *Algorithm 6 has complexity $O(n + m)$.*

Proof. Computing a MNS cocomp ordering can be done in $O(n + m)$ using $LBFS^+$ or MCS^+(see chapter 1). *ChainClique* have a space and time complexity $O(n + m)$. Enumerate the cliques is in $O(n + m)$. Computing first and last can be done by enumerating all the cliques of the spanning interval graph. The computation of forward, backward can be done by enumerating for each vertex its neighbourhood after computing first and last an so can be done in $O(n + m)$. Enumerating the vertices can be done in $O(n)$.

□

4.4 Minimal clique separators in cocomparability graphs

This section is organized as follows. In the first subsection, we present the problem of finding minimal clique separators and our algorithm. The second subsection is devoted to prove some results about clique separators in cocomparability graphs. The third subsection is devoted to the study of the structure of the atoms of a cocomparability graph. The last subsection is devoted to the correctness of the algorithm.

4.4.1 The problem of minimal clique separators

We recall that for a connected graph G, $S \subseteq V(G)$ is a **separator** if $G - S$ is disconnected. For two vertices $a,\ b \in V(G)$, S is an ab-separator if a and b are in different connected components of $G - S$. S is a minimal ab-separator if no proper subset of S is an ab-separator. S is a **minimal clique separator** of a graph G if S is a clique and there is some vertex a, b such that S is a minimal ab-separator. An atom is a maximal induced subgraph that admits no clique separator.

Traditionally the decomposition is presented using a tree. Each node is a clique separator and the leaves formed the connected components. For the decomposition to be unique, the clique separators must be minimal clique separators. For an example, see Figure 4.2.

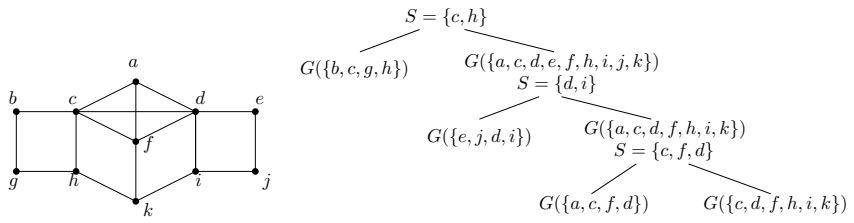


Figure 4.2: A graph G and its decomposition tree using minimal clique separators.

In the tree decomposition, the atoms are exactly the leaves of the tree. There is always at most n atoms and the size of the decomposition is in $O(n + m)$ (see

[BPS10]) .

Our algorithm to decompose a cocomparability graph using its clique separators does not output the tree decomposition. But instead, it outputs the list of the atoms of the graph: $A_1, \cdots, A_l$ such that $A_a \cap A_c \subseteq A_b$ for $1 \leq a \leq b \leq c \leq l$ and $A_a \cap A_b$ is a clique. The advantage is it is simpler to handle than a tree decomposition but still allows us to use the divide and conquer strategy to solve a problem. To get the clique separators, we just have to compute the intersection of two consecutive atoms in the list.

For example, for the coloring problem the algorithm is:

Algorithm 7: Divide and conquer

Data: The atoms A_1, A_l such that $A_a \cap A_c \subseteq A_b$ for $1 \leq a \leq b \leq c \leq l$ and $A_a \cap A_b$ is a clique

Result: An optimal coloring

```
i ← 1;
while i ≤ l do
    Find an optimal coloring for A_i;
    Join it to the current coloring;
    i ← i + 1;
end
Output the coloring;
```

The correctness of this algorithm relies on the interval structure of the atoms. We know that the intersection of two atoms forms a clique and that $A_a \cap A_c \subseteq A_b$ for $1 \leq a \leq b \leq c \leq l$. Interval graphs have the same structure and so the proof is the same as for interval graphs, except that the atoms are complete graphs in the case of interval graphs. Since we assume that we have a procedure to perfectly compute an optimal coloring of an atom, the algorithm will ouput an optimal coloring of the graph.

The same kind of algorithm works for the maximum clique problem and the minimum fill-in. For the maximum independent set, in cocomparability graphs

the clique decomposition does not help much since a greedy algorithm on a special cocomp ordering works (see [CDHK]).

We now present the algorithm to compute a decomposition of cocomparability graphs using clique separator (Algorithm 8). The correctness of the algorithm will mainly rely on theorem 4.4.7, theorem 4.4.8 and theorem 4.2.1.

As for the simplicial vertices, we will also compute first, last, backward and forward. C_a^i will be the set of vertices that appear or have some neighbor that appears before the clique C_i in the interval graph and C_b^i will be the set of vertices that appear or have a neighbor that appears after the clique C_i in the interval graph.

For an example, let us consider the graph in Figure 4.3. $\sigma = v_1, v_2, v_3, v_4, v_6, v_5, v_7, v_9, v_8$ is a LBFS cocomp ordering. Chainclique outputs $\{v_1, v_2, v_3\}$, $\{v_2, v_4\}$, $\{v_2, v_6\}$, $\{v_5, v_6, v_7\}$, $\{v_6, v_9\}$, $\{v_9, v_8\}$. Algorithm 8 will output $\{v_1, v_2, v_3\}$, $\{v_2, v_3, v_4, v_5, v_6\}$, $\{v_5, v_6, v_7\}$ and $\{v_5, v_6, v_8, v_9\}$.

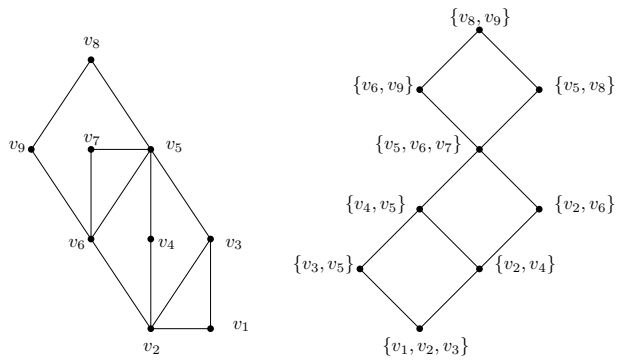


Figure 4.3: A cocomparability graph and its lattice.

4.4.2 Clique separators and components in cocomparability graphs

This subsection is organized as follows. First, we recall a well-known proposition and introduce some terminology, then we prove some claims, lemma and theorem on the structure of the clique separators in cocomparability graphs.

Algorithm 8: List of atoms

Data: a connected graph $G = (V(G), E(G))$ and a MNS cocomp ordering τ of $V(G)$

Result: A sequence of the atoms of G

```
C_1, ..., C_k = Chainclique(G, τ);
Compute first, last, backward, forward for every vertex of V;
Let L = ∅ %{The sequence of the atoms}%;
Let Dis ← ∅;
Let A ← ∅;
for (i ← 1 to k) do
    if (∄x ∈ Dis and forward[x] > i) then
        Let C_a^i ← {u ∈ C_i | first[u] < i or backward[u] < i};
        Let C_b^i ← {u ∈ C_i | last[u] > i or forward[u] > i};
        if C_a^i ≠ ∅ and (A ⊈ C_a^i) then
            Append (A ∪ C_a^i) to L;
        end
        if (C_i − C_a^i ≠ ∅ and C_i − C_b^i ≠ ∅) then
            Append (C_i) to L;
        end
        A ← C_b^i;
    else
        A ← A ∪ {v ∈ C_i | last[v] = i};
    end
    Dis ← Dis ∪ {u ∈ C_i | last[u] = i};
end
Output L;
```

Proposition 4.4.1. *Let G be a graph.*

S is a minimal separator for two connected components D_1, D_2 of $G - S$ if and only if $\forall x \in S$, $N(x) \cap D_1 \neq \emptyset$ and $N(x) \cap D_2 \neq \emptyset$.

Let G be a cocomparability graph and σ a cocomp ordering. For a maximal clique A, let us denote by $BELOW(A) = \{x \in V|$ there exists a maximal clique C_x, $x \in C_x$ and $C_x <_{AM(P_\sigma)} A\}$ and $ABOVE(A) = \{x \in V|$ there exists a

maximal clique C_x, $x \in C_x$ and $A <_{AM(P_\sigma)} C_x\}$. We will say that S is a minimal separator for two components D_1, D_2 if there exists $u \in D_1$, $v \in D_2$ such that S is a minimal uv-separator.

Claim 4.4.2. *Let G be a cocomparability graph, let σ be a cocomp ordering, let S be a clique separator of G and let $D_1, \ldots, D_l$ be the connected components of $G - S$.*

For all connected components D_i, D_j of $G - S$ such that $D_i \neq D_j$ either $\forall x \in D_i$, $\forall y \in D_j$, $x <_\sigma y$ or $\forall x \in D_i$, $\forall y \in D_j$, $y <_\sigma x$.

Proof. Assume for contradiction that for some connected components D_i and D_j the property is not true. Then either $\exists u, v \in D_i, \exists x \in D_j$ such that $u <_\sigma x <_\sigma v$ or $\exists u, v \in D_j, \exists x \in D_i$ such that $u <_\sigma x <_\sigma v$. Assume we are in the first case. Let $u = y_1, \ldots, y_g = v$ be an induced path in D_i. Such a path has to exist since D_i is a connected component. Let y_a be the rightmost vertex of the path such that $y_a <_\sigma x$ and so $x <_\sigma y_{a+1}$. But since x is not in the connected component D_i, we have that $y_a x$, $y_{a+1} x \notin E$. Therefore the umbrella $y_a <_\sigma x <_\sigma y_{a+1}$ contradicts that σ is a cocomp ordering. The argument for the second case is the same, which again gives us a contradiction. □

The previous claim shows that with a cocomp ordering we can define a total order on the connected components of $G - S$. So let σ be a cocomp ordering and for a minimal clique separator S let $D_1^\sigma, \ldots, D_k^\sigma$ be the connected components such that if $i < j$ then $\forall x \in D_i^\sigma$, $\forall y \in D_j^\sigma$, $x <_\sigma y$.

Claim 4.4.3. *If S is a minimal clique separator for the connected components D_i^σ, D_j^σ then for every connected components D_g^σ, $i < g < j$, S is universal to D_g^σ and S is a minimal clique separator for the connected components D_i^σ, D_g^σ and for the components D_j^σ, D_g^σ.*

Proof. Assume for contradiction that there exists $u \in D_g^\sigma$, $v \in S$ such that $uv \notin E$. Since S is a minimal clique separator for D_i^σ, D_j^σ there exists $x \in D_i^\sigma$, $y \in D_j^\sigma$ such that $xv \in E$ and $yv \in E$. From the ordering of the components we have $x <_\sigma u <_\sigma y$. Now we have two cases: either $v <_\sigma u$ or $u <_\sigma v$. In the first case v, u, y is an umbrella, which contradicts σ to be a cocomp ordering. In the second case x, u, v is an umbrella, which again contradicts σ to be a cocomp ordering.

Since S is a minimal clique separator for the components D_i^σ, D_j^σ let $u \in D_i^\sigma$, $v \in D_j^\sigma$ such that S is a minimal uv-separator. Let x be any vertex of D_g. Since x is universal to S, S is a minimal ux-separator and a minimal xv-separator. □

Let S be a clique separator of G. We start by remarking that a maximal clique of G is include in $S \cup D_h^\sigma$ for some connected component D_h^σ. The next lemmas are going to be useful in the proofs.

Lemma 4.4.4. *Let G be a cocomparability graph, let σ be a cocomp ordering, let S be a minimal clique separator of G and let $D_1^\sigma, \ldots, D_k^\sigma$ be the connected components of $G - S$.*

If A, B are some maximal cliques of G, $A \subseteq D_i^\sigma \cup S$, $B \subseteq D_j^\sigma \cup S$, $i < j$ then $A \leq_{MA(P_\sigma)} B$.

Proof. Let us prove that for every $y \in B$, there exists $x \in A$ such that $x \leq_{P_\sigma} y$. We have two cases: either $y \in B \cap A$ or $y \in B - A$. In the first, since $y \leq_{P_\sigma} y$, we have the property. Now in the second case, since $y \in B - A$ and A is a maximal clique, there exists $x \in A$ such that $xy \notin E$. Here we have again two cases: either $x \in D_i^\sigma$ or $x \in S$.

In the first case, we have again two cases: either $y \in D_j^\sigma$ or $y \in S$. In the first case, from the assumption on the ordering of the components, we know $x <_\sigma y$ and so $x \leq_{P_\sigma} y$. In the second case, assume for contradiction that $y <_\sigma x$. Since S is a minimal clique separator there exists a neighbor z of y in D_j^σ. From the ordering of the components we know that $x <_\sigma z$. But now y, x, z is an umbrella, which contradicts the fact that σ is a cocomp ordering. Thus $x \leq_{P_\sigma} y$.

In the second case, since S is a clique, $xy \notin E$ and $x \in S$ we see that $y \notin S$ and so $y \in D_j^\sigma$. Now assume for contradiction that $y <_\sigma x$. Since S is a minimal clique separator there exists a neighbor z of x in D_i^σ. From the ordering of the components we know that $z <_\sigma y$. But now z, y, x is an umbrella which contradicts the fact that σ is a cocomp ordering. Thus $x <_\sigma y$ and so $x \leq_{P_\sigma} y$.

Therefore since for every $y \in B$, there exists $x \in A$ such that $x \leq_{P_\sigma} y$, we deduce that $A \leq_{MA(P_\sigma)} B$. □

Lemma 4.4.5. *Let G be a cocomparability graph and σ a cocomp ordering of G.*

If A, B, C are three maximal fully comparable clique such that $A <_{MA(P_\sigma)} B <_{MA(P_\sigma)} C$, then $G - B$ is not connected.

Proof. Let $D_1 = \{x \in V | \exists C_x \text{ s.t. } x \in C_x \text{ and } C_x <_{AM(P)} B\} - B$ and $D_2 = \{x \in V | \exists C_x \text{ s.t. } x \in C_x \text{ and } B <_{AM(P)} C_x\} - B$. Assume for contradiction that D_1 and D_2 are connected.

First, let prove that $D_1 \cap D_2 = \emptyset$. Assume for contradiction that $D_1 \cap D_2 \neq \emptyset$ and let $u \in D_1 \cap D_2$. From our definition of D_1, we know that there exists C_u such that $u \in C_u$ and $C_u <_{MA(P_\sigma)} B$. From our definition of D_2, we know that there exists C'_u such that $u \in C'_u$ and $B <_{MA(P_\sigma)} C'_u$. Using proposition 2.3.3 on $C_u,\ B,\ C'_u$ we deduce that $u \in B$. But now u cannot belong to D_1 or D_2 since we remove the vertices of B from D_1 and D_2. Therefore $D_1 \cap D_2 = \emptyset$.

Since we assume D_1 and D_2 to be connected and $D_1 \cap D_2 = \emptyset$, we know that there exists $u \in D_1$, $v \in D_2$, $u \neq v$ and $uv \in E(G)$. Let C_u be a maximal clique such that $C_u < B$, let C_v be a maximal clique such that $B < C_v$ and C_{uv} be a maximal clique such that $u,\ v \in C_{uv}$. Since B is fully comparable, we have either $C_{uv} < B$ or $B < C_{uv}$. In the first case, using proposition 2.3.3 on $C_{uv},\ B,\ C_v$, we deduce that $v \in B$, which is a contradiction to $v \in D_2$. In the second case, using proposition 2.3.3 on $C_u,\ B,\ C_{uv}$, we deduce that $u \in B$, which is a contradiction to $u \in D_1$. Therefore D_1 and D_2 are not connected and so B is a clique separator. □

In the next theorem, we exhibit the structure of the components.

Theorem 4.4.6. *Let G be a cocomparability graph, let σ be a cocomp ordering, let S be a clique separator of G and $D_1^\sigma, ..., D_l^\sigma$ be the connected components ordered by σ.*

For every connected component D_i^σ of $G - S$ such that $D_i^\sigma \cup S$ is not a complete graph, there exists two different maximal cliques C_a, C_b, $C_a <_{MA(P_\sigma)} C_b$ such that:

- *C_a and C_b are fully comparable*
- *$D_i^\sigma \cup S = ABOVE(C_a) \cap BELOW(C_b)$.*

Proof. Let Q be the subposet of P_σ induced by the set $D_i^\sigma \cup S$. Since $D_i^\sigma \cup S$ is not a complete graph, Q has height at least one and there exists C_a as the set of sources of Q distinct from C_b, the set of sinks of Q. Clearly they are both maximal clique of G.

Let us show that C_a and C_b are fully comparable. Using lemma 4.4.4, every maximal clique B of G not included in $D_i^\sigma \cup S$ must be comparable to C_a and C_b.

Now for any maximal clique B included in $D \cup S$, since C_a and C_b are respectively the set of sources and the set of sinks, we must also have that B is comparable to C_a and C_b. Therefore we deduce that they are fully comparable.

Let us now show that $D_i^\sigma \cup S = ABOVE(C_a) \cap BELOW(C_b)$. Let first show that $D_i^\sigma \cup S \subseteq ABOVE(C_a) \cap BELOW(C_b)$. For every vertex $x \in D_i^\sigma \cup S$, there exists a maximal clique $C_x \subseteq D \cup S$ such that $x \in C_x$. Since C_a and C_b are fully comparable and are respectively the sets of sources and sinks of Q, we deduce that $C_a \leq_{MA(P)} C_x \leq_{MA(P)} C_b$. Therefore $D \cup S \subseteq ABOVE(C_a) \cap BELOW(C_b)$. Let us now show that $ABOVE(C_a) \cap BELOW(C_b) \subseteq D \cup S$. Assume for contradiction that $ABOVE(C_a) \cap BELOW(C_b) \nsubseteq D_i^\sigma \cup S$ and let $u \in (ABOVE(C_a) \cap BELOW(C_b)) - (D \cup S)$. Therefore there exists a maximal clique C_u such that $C_a <_{MA(P)} C_u <_{MA(P)} C_b$. Since $u \notin D_i^\sigma \cup S$, assume $u \in D_j^\sigma$ with $i \neq j$. We now have two cases: $i < j$ and $j < i$. In the first case, using lemma 4.4.4 on C_u and C_a, we deduce that $C_u <_{MA(P)} C_a$, which contradicts $C_a <_{MA(P)} C_u$. In the second case, using lemma 4.4.4 on C_u and C_b, we deduce that $C_b <_{MA(P)} C_u$, which contradicts $C_u <_{MA(P)} C_b$.

□

4.4.3 Structure of the atoms of a cocomparability graph

We now exhibit the structure of the atoms in a cocomparability graph.

Theorem 4.4.7. *Let G be a cocomparability graph, let σ be a cocomp ordering and let A be a subset of $V(G)$ and A is not a clique.*

A is an atom if and only there exists two maximal cliques fully comparable $C_a <_{MA(P_\sigma)} C_b$ such that:

- *There is no fully comparable maximal clique D such that $C_a <_{MA(P_\sigma)} D <_{MA(P_\sigma)} C_b$*
- $A = ABOVE(C_a) \cap BELOW(C_b)$

Proof. For the forward direction, let S_1 be a minimal clique separator of G and $D^\sigma_{(1,1)}, \cdots, D^\sigma_{(1,k_1)}$ the set of connected components of $G - S_1$. Let $D^\sigma_{(1,j_1)}$ be the connected component such that $A \subseteq D^\sigma_{(1,j_1)} \cup S_1$. Using theorem 4.4.6 on $D^\sigma_{(1,j_1)}$, we know that there exists C_a^1 and C_b^1 such that C_a^1 and C_b^1 are fully

comparable and $D^{\sigma}_{(1,j_1)} \cup S_1 = ABOVE(C^1_a) \cap BELOW(C^1_b)$. Now we have two cases: either $D^{\sigma}_{(1,j_1)} \cup S_1 = A$ or $D^{\sigma}_{(1,j_1)} \cup S_1 \neq A$. In the first case, since A is an atom there must not exist a fully comparable maximal clique D such that $C^1_a <_{MA(P_\sigma)} D <_{MA(P_\sigma)} C^1_b$ otherwise using lemma 4.4.5 we deduce that A admits a clique separator, which contradicts the fact that A is an atom. Therefore, in the case $D^{\sigma}_{(1,j_1)} \cup S_1 = A$ the theorem is true. In the second case, since A is an atom and $D^{\sigma}_{(1,j_1)} \cup S_1 \neq A$, there must exists a minimal clique separator that disconnects $D^{\sigma}_{(1,j_1)} \cup S_1$ into a series of connected components $D^{\sigma}_{(1,2)}, \cdots, D^{\sigma}_{(1,k_2)}$. Now we are back in the original situation and we can use the same argument. By keeping using the same argument and since G is finite, we reach the conclusion that the theorem is true.

Conversely, assume for contradiction that A is not an atom. We have two cases: either there exists a clique separator in A or A is a strict subset of an atom.

In the first case, let S be a clique separator of A and $D^{\sigma}_1, ..., D^{\sigma}_l$ the connected components of $A - S$. We now have again two cases: either every connected components $D^{\sigma}_i \cup S$ is a clique or there exists $D^{\sigma}_j \cup S$ which is not a clique. In the first case, we deduce that $C_a = D^{\sigma}_1 <_{MA(P_\sigma)} ... <_{MA(P_\sigma)} D^{\sigma}_l = C_b$. Therefore, we must have $l = 2$ otherwise we contradict that there is no fully comparable maximal clique D such that $C_a <_{MA(P_\sigma)} D <_{MA(P_\sigma)} C_b$. So we have C_b that covers C_a. Since C_a and C_b are fully comparable, we further get that $ABOVE(C_a) \cap BELOW(C_b) \subseteq C_b$, which is a contradiction to A not being a clique. In the second case, using theorem 4.4.6 on D^{σ}_j we deduce that there exists two maximal fully comparable clique C^1_a, C^1_b such that $D^{\sigma}_1 \cup S = ABOVE(C^1_a) \cap BELOW(C^1_b)$. But now using the assumption we deduce that $C_a \leq_{MA(P_\sigma)} C^1_a <_{MA(P_\sigma)} C^1_b \leq_{MA(P_\sigma)} C_b$. Since $D^{\sigma}_1 \cup S$ is a strict subset of A, $A = ABOVE(C_a) \cap BELOW(C_b)$ and $D^{\sigma}_1 \cup S = ABOVE(C^1_a) \cap BELOW(C^1_b)$, we further deduce that $C_a \neq C^1_a$ or $C^1_b \neq C_b$. In both cases, we contradict the assumption that there is no fully comparable maximal clique D such that $C_a <_{MA(P_\sigma)} D <_{MA(P_\sigma)} C_b$.

In the second case, let B be an atom of G such that $A \subsetneq B$. Using the forward direction, we deduce that there exists two fully comparable maximal cliques C^1_a and C^1_b such that there is no fully comparable maximal clique D such that $C^1_a <_{MA(P_\sigma)} D <_{MA(P_\sigma)} C^1_b$ and $B = ABOVE(C^1_a) \cap BELOW(C^1_b)$. Since

$A \subseteq B$, we deduce that $C_a^1 \leq_{MA(P_\sigma)} C_a <_{MA(P_\sigma)} C_b \leq_{MA(P_\sigma)} C_b^1$. Since A is a strict subset of B, $B = ABOVE(C_a^1) \cap BELOW(C_b^1)$ and $A = ABOVE(C_a) \cap BELOW(C_b)$ we further deduce that $C_a \neq C_a^1$ or $C_b^1 \neq C_b$. In both cases, we contradict the assumption that there is no fully comparable maximal clique D such that $C_a^1 <_{MA(P_\sigma)} D <_{MA(P_\sigma)} C_b^1$.

□

Theorem 4.4.8. *Let G be a cocomparability graph, let σ be a cocomp ordering and let A be a clique of $V(G)$.*

A is an atom of G if and only if A is a maximal fully comparable clique, $A \not\subseteq BELOW(A)$ and $A \not\subseteq ABOVE(A)$.

Proof. ($\Rightarrow$) For the forward direction, first we show that if A is an atom then A must be a maximal clique. Then we show that A is fully comparable. Finally we show that $A \not\subseteq BELOW(A)$ and $A \not\subseteq ABOVE(A)$.

Assume for contradiction that A is an atom and A is not a maximal clique. Let u be an universal vertex to A. Since A is an atom, $A \cup \{u\}$ is not an atom. Since $A \cup \{u\}$ is a clique, $A \cup \{u\}$ does not admit a clique separator. Therefore there must exist an atom B such that $A \cup \{u\} \subseteq B$. But now this is a contradiction to the fact that A is an atom. Hence, A must be a maximal clique.

In order to show that A is fully comparable, we will use the same kind of argument as in the previous theorem. Let S_1 be a minimal clique separator of G and $D^\sigma_{(1,1)}, \cdots, D^\sigma_{(1,k_1)}$ the set of connected components of $G - S_1$. Let $D^\sigma_{(1,j_1)}$ be the connected component such that $A \subseteq D^\sigma_{(1,j_1)} \cup S_1$. We have two cases: either $A = D^\sigma_{(1,j_1)} \cup S_1$ or $A \neq D^\sigma_{(1,j_1)} \cup S_1$. In both cases, using lemma 4.4.4 on A and on any maximal clique not included in $D^\sigma_{(1,j_1)} \cup S_1$, we deduce that A is comparable with any maximal clique not included in $D^\sigma_{(1,j_1)} \cup S_1$. Therefore, in the first case, we deduce that A is fully comparable. In the second case, since A is an atom and $A \subsetneq D^\sigma_{(1,j_1)} \cup S_1$ we deduce that there must exist S_2 such that $S_2 \subseteq D^\sigma_{(1,j_1)} \cup S_1$ and S_2 is a clique separator. Hence we can repeat the same arguments. By keeping doing that since the graph is finite we end up deducing that A is fully comparable.

Let us now show that $A \not\subseteq BELOW(A)$ and $A \not\subseteq ABOVE(A)$. Assume for contradiction that $A \subseteq BELOW(A)$. Let B be the maximal fully comparable clique such that $B <_{MA(P_\sigma)} A$ and $\nexists D$ fully comparable such that $B <_{MA(P_\sigma)}$

$D <_{MA(P_\sigma)} A$. Now we have two cases: either $BELOW(A) \cap ABOVE(B)$ is a clique or not. In the first case, since $A \subseteq BELOW(A)$ we must have $A = BELOW(A) \cap ABOVE(B)$ otherwise we contradict the fact that A is a maximal clique. Since $A = BELOW(A) \cap ABOVE(B)$, any maximal clique D such that $B <_{MA(P_\sigma)} D <_{MA(P_\sigma)} A$ is a subset of A which contradicts either that $D \neq A$ or D is a maximal clique. Hence A covers B and since $A = BELOW(A) \cap ABOVE(B)$, from the definition of $BELOW$ we deduce that A is a subset of B which contradicts either that A is a maximal clique or $B \neq A$. In the second case, using the previous theorem we deduce that $BELOW(A) \cap ABOVE(B)$ is an atom and $A \subsetneq BELOW(A) \cap ABOVE(B)$ which contradicts the assumption that A is an atom. The proof for $A \not\subseteq ABOVE(A)$ is completely symmetric.

($\Leftarrow$) Conversely, assume that A is a maximal fully comparable clique, $A \not\subseteq BELOW(A)$, $A \not\subseteq ABOVE(A)$ and A is not an atom. Since A does not admit a clique separator, let B be the atom of G such that $A \subseteq B$. Using the previous theorem, we deduce that there exists two maximal clique C_a and C_b such that there is no fully comparable maximal clique D such that $C_a <_{MA(P_\sigma)} D <_{MA(P_\sigma)} C_b$ and $A = ABOVE(C_a) \cap BELOW(C_b)$. Since A is fully comparable and there is no fully comparable maximal clique D such that $C_a <_{MA(P_\sigma)} D <_{MA(P_\sigma)} C_b$, we deduce that either $A = C_a$ or $A = C_b$. In the first case, since $A = ABOVE(C_a) \cap BELOW(C_b)$ we deduce that $A \subseteq ABOVE(A)$, which contradicts the assumption $A \not\subseteq ABOVE(A)$. In the second case, since $A = ABOVE(C_a) \cap BELOW(C_b)$ we deduce that $A \subseteq BELOW(A)$, which contradicts the assumption $A \not\subseteq BELOW(A)$.

□

4.4.4 Correctness of the Algorithm

We have now finished with the structure of the atoms and turn our attention to the correctness of the algorithm.

Claim 4.4.9. *Let C_i be a clique such that $C_i \in Chainclique(G, \sigma)$.*

If C_i is not a maximal clique then there exists vertices x, y such that $xy \in E$, $last[y] < i$ and $first[x] > i$.

Proof. Let us assume that C_i is not a maximal clique; let v be the first vertex of $C_i - C_{i-1}$ in σ.

If it exists, let w be the rightmost vertex complete to C_i, $w \notin C_i$ and $w <_\sigma v$. Let $C_h = C_{last[w]}$. We have that $h < i$. Since w is a neighbour of v and w is not in C_i, we know that $w \notin C_i$ and thus $h < i - 1$. So now let us consider the first vertex x of $C_{h+1} - C_h$ in the ordering σ. Because w doesn't belong to C_{h+1} we know that $wx \notin E$ and because w is universal to C_i we have that $x \notin C_i$. Because w is the rightmost vertex complete to C_i, x must not be adjacent to some vertex y of C_i. Now either $x <_\sigma y$ or $y <_\sigma x$. In the first case we know that $w <_\sigma x <_\sigma y$ is an umbrella and so σ is not a cocomp ordering which is a contradiction. In the second case since y appears before x, we have that $first[y] \leq h + 1$. But now y belongs to $C_{first[y]}$ and C_i and so using property 3.2.2, we know that $y \in C_{h+1}$ which is a contradiction to $xy \notin E$. Thus there is no vertex complete to C_i that appears before v.

Now let w be the leftmost complete vertex to C_i, $w \notin C_i$ and $v <_\sigma w$. Let x be the first vertex of $C_{i+1} - C_i$ and so $x <_\sigma w$, $first[x] = i+1$. At the time where x has been chosen by MNS, since x is not universal to C_i and w is universal to C_i, x must be adjacent to a vertex y that has already been visited and $yw \notin E$. Since $yw \notin E$, we see $y \notin C_i$. Since y has been visited before x we deduce $last[y] < i$. □

Claim 4.4.10. *Let C_i be a maximal clique such that $C_i \in Chainclique(G, \sigma)$.*

C_i admits a maximal incomparable clique in $MA(P_\sigma)$ if and only if there exists vertices u, v such that $uv \in E$, $last[u] < i$ and $first[v] > i$.

Proof. For the forward direction assume that C_i admits an incomparable clique. Let D be a clique such that $D \parallel_{MA(P_\sigma)} C_i$. Since $D \parallel_{MA(P_\sigma)} C_i$, $S^-(D, C_i) \neq \emptyset$ and $S^+(D, C_i) \neq \emptyset$ otherwise from the definition of $\vee_{MA(P^\sigma)}$ and $\wedge_{MA(P^\sigma)}$ we deduce that either $D \subseteq D \vee_{MA(P_\sigma)} C_i$ or $D \subseteq D \wedge_{MA(P_\sigma)} C_i$, therefore contradicting the maximality of D or $D \parallel_{MA(P_\sigma)} C_i$. Let $u \in S^-(D, C_i)$ and $v \in S^+(D, C_i)$. Since $u \in S^-(D, C_i)$, there exists $x \in C_i$ such that $u <_{P_\sigma} x$. Since σ is a linear extension of P_σ we see that $u <_\sigma x$. Since $ux \notin E$ we see $u \notin C_i$. Since $u <_{P_\sigma} x$, $u \notin C_i$ we deduce that $last[u] < i$. The same argument gives us that $first[v] > i$.

For the reverse direction assume there exists u, v such that $uv \in E$, $last[u] < i$ and $first[v] > i$. Let us prove that since $last[u] < i$ there exists a maximal clique C_u of G such that $u \in C_u$ and $C_u <_{MA(P_\sigma)} C_i$. Assume for contradiction that there is no maximal clique C_u. Let C_u^0 be a maximal clique such that $u \in C_u^0$.

Using proposition 2.3.4, either $u \in C_u^0 \wedge C_i$ or $u \in C_u^0 \vee C_i$. In the first case we contradict our assumption that there is no maximal clique such that u belongs to it and is less than C_i in $MA(P_\sigma)$. In the second case, since C_i is a maximal clique and u does not belong to C_i there exists $v \in C_i - N(u)$. Since $last[u] < i$ we must have that $u <_\sigma v$. But using lemma 2.3.1 on C_i and $C_u^0 \vee C_i$ we deduce that $v <_{P_\sigma} u$. Therefore it contradicts that σ is a linear extension of P_σ. By doing the same we can deduce that there exists a maximal clique C_v such that $v \in C_v$ and $C_i <_{MA(P_\sigma)} C_v$. Using claim 3.4.2, we deduce that there exists a maximal clique C_{uv} such that $C_u <_{MA(P_\sigma)} C_{uv} <_{MA(P_\sigma)} C_v$. Now we have three cases: $C_{uv} <_{MA(P_\sigma)} C_i$; $C_i <_{MA(P_\sigma)} C_{uv}$; $C_i \parallel_{MA(P_\sigma)} C_{uv}$. In the first case, since $v \in C_{uv}$, $v \in C_v$, proposition 2.3.3 implies that $v \in C_i$ which contradicts that $first[v] > i$. So this case can't happen. In the second case, since $u \in C_{uv}$, $u \in C_u$, proposition 2.3.3 implies that $u \in C_i$ which contradicts that $last[u] < i$. So this case can not happen. In the last case we deduce that there exists an incomparable clique to C_i. □

Theorem 4.4.11. *Let $G = (V(G), E(G))$ be a cocomparability graph and σ a MNS cocomp ordering of the vertices of G.*

Algorithm 8 outputs all the atoms of G and has complexity in $O(n + m)$.

Proof. Theorem 4.2.1 guarantees that any fully incomparable clique will belong to $C_1, ..., C_k$. The two claims 4.4.10 and 4.4.9 confirm that our algorithm will detect when a clique is fully comparable. Theorem 4.4.8 certifies that if G has a atom which is a clique, then this atom will be found by the algorithm. Theorem 4.4.7 certifies that if G has a atom which is not a clique then the atom will be found by the algorithm.

Computing a MNS cocomp ordering can be done in $O(n + m)$ using for example $LBFS^+(G, \sigma)$ where σ is a cocomp order. The ChainClique algorithm has complexity $O(n+m)$. Enumerating all the cliques can be done in $O(n+m)$. Computing first and last can be done by enumerating all the cliques of the spanning interval graph. The computation of forward, backward can be done by enumerating for each vertex its neighbourhood after computing first and last and so can be done in $O(n + m)$. To verify that $\nexists x \in Dis$ and $forward[x] > i$, we just have to keep the $x \in Dis$ such that $\forall y \in Dis$, $forward[x] \geq forward[y]$ and so can be done easily. Now at each step, we can easily check all the conditions in $O(C_i)$.

Therefore the algorithm has complexity $O(n + m)$.

□

4.5 Conclusions and perspectives for further work

The most interesting result of this section is that we have a new way to decompose a partial order. This decomposition can be found efficiently and has good algorithmic perspectives. One interesting application is to find an optimal coloring for cocomparability graphs. So far the best algorithm use matrix multiplication. We can ask that if we use the clique separators as a preliminaries step, what are the consequences on the problem of coloring cocomparability graphs? The main problem is that a cocomparability graph may not have a clique separator (for example complete bipartite graphs). So it raises lot of combinatoric questions: What is the probability of having a clique separator? Given a partial order, what is the maximum size of an atom?

An other way to use it is to define new graph classes; for example a cocomparability graph whose atoms are all cobipartite graphs. This class can be recognized in linear time using our algorithm to find the decomposition and then test if each atom is a cobipartite graph. For this class, we do not even need the decomposition algorithm and can use the fully comparable clique, testing if between two fully comparable cliques the graph forms a cobipartite graph.

4. SIMPLICIAL VERTICES AND CLIQUE SEPARATORS IN COCOMPARABILITY GRAPHS

5

Transitive Orientation: Multisweep LBFS

5.1 Introduction

In this chapter, we present an algorithm to find a cocomp ordering. Therefore, it can also be seen as an algorithm to find a transitive orientation. There already exists several algorithms to find a co-umbrella free ordering of a comparability graph, including a linear time one described in [MS99] but it is quite involved. There exists also algorithms with running time $O(n + mlogn)$ like the one presented in [HMPV00], which are easier to handle. It should be noticed that these algorithms do not provide a fast (i.e. linear) way to recognize comparability graphs and cocomparability graphs. The classic strategy to recognize a cocomparability (resp. comparability) graph is by using first an algorithm to find a umbrella (resp. co-umbrella) free ordering and then check if this ordering is umbrella (resp. co-umbrella) free. Up to now, it is still unknown if we can perform this testing operation in linear time. So far the best algorithm for testing this ordering uses boolean matrix multiplication [Spi03].

The algorithm we present in this chapter only uses LBFS$^+$. In the worst case, it performs n LBFS$^+$. This kind of algorithm is often referred as a multisweep algorithm. This algorithm answers positively a question asked in [COS09], namely : whether there exists a multisweep algorithm based on LBFS to find a cocomp ordering of a cocomparability graph. This algorithm has running time $O(nm)$ and even if the algorithm is slower than the one in [MS99], for the problem of recognition it will perform as well as the other ones with complexity less than

$O(nm)$. The strong point of this algorithm lies in its simplicity. Furthermore, this algorithm gives some hope for the existence of a simple multisweep "LBFS" based algorithm with linear running time to transitively orient comparability graphs (see 5.4.1).

This chapter is organized as follows. In section 5.2, we present the algorithm. Section 5.3 is devoted to the correctness of the algorithm. Section 5.4 contains conclusion and some conjectures.

This work has been done in collaboration with Michel Habib and submitted [DH].

5.2 Series of LBFS

Let us start by introducing the **Repeated LBFS$^+$** algorithm which is the main algorithm studied in this chapter.

Algorithm 9: Repeated LBFS$^+$

Data: $G = (V(G), E(G))$ an undirected graph
Result: an ordering σ_n

```
σ_1 ← LBFS(G);
for i = 2 to |V(G)| do
    σ_i ← LBFS+(G,σ_{i-1});
end
Output σ_{|V(G)|};
```

Let us illustrate the behaviour of **Repeated LBFS$^+$** with the graph on figure 5.1. The first ordering, σ_1 will be for example $\sigma_1 =< d, c, b, f, a, e >$. Therefore $\sigma_2 =< e, f, b, d, c\ a >$, $\sigma_3 =< a, c, b, d, e, f >$, $\sigma_4 =< f, e, d, b, c, a >$, $\sigma_5 =< a, c, b, d, e, f >= \sigma_3$, $\sigma_6 =< f, e, d, b, c, a >= \sigma_4$. It is easy to check that σ_3 and σ_4 have no umbrella.

This chapter is devoted to a proof of the following result:

Main result: G is a cocomparability graph if and only if Repeated LBFS$^+$ computes a cocomp ordering.

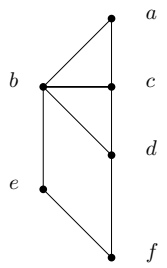


Figure 5.1: A cocomparability graph $G = (V(G), E(G))$

The following result has already been proved for interval graphs but has not been published.

Theorem 5.2.1. *[CK] $G = (V(G), E(G))$ is an interval graph if and only if Repeated LBFS$^+$ computes an interval ordering.*

We now generalize it here for the wider class of cocomparability graphs. In corollary 5.3.16, we will show that the previous theorem is a corollary of our result. Since unfortunately there are examples on which the convergence depends on the number of vertices, see [Ma, COS09], Repeated$^+$ cannot use a constant number of LBFS$^+$.

The use of LBFS is also motivated by the two following results.

Theorem 5.2.2. *[HMPV00] Let s be the last vertex visited by a LBFS on a cocomparability graph G; then there exists a cocomp ordering starting (resp. ending) at s.*

Therefore the last vertex of a LBFS in a cocomparability graph can be taken as the starting point of a cocomp ordering and so these two results give us some hope that a series of successive LBFS$^+$ produces a cocomp ordering.

5.3 Proof of Repeated LBFS$^+$ Algorithm

5.3.1 Sketch of the proof of our main result

Our proof that Repeated LBFS$^+$ computes a cocomp ordering is quite involved and as already mentioned they are not so many tools to analyze multisweep algo-

rithms. We propose an algorithmic proof by introducing a companion algorithm Partition algorithm to LBFS+, and studying their common invariants in order to prove that they both produce a cocomp ordering (not necessarily the same one).

This requires particular notation and definitions which are introduced in the next subsection.

5.3.2 Definitions and notations

Let $\mathcal{X} =< \mathcal{X}(1), ..., \mathcal{X}(k) >$ be an ordered partition of $V(G)$, in which $\mathcal{X}(j)$ denotes the jth part of the partition $\mathcal{X}$. $|\mathcal{X}|$ denotes the number of parts of the partition $\mathcal{X}$. Let $\mathcal{X}^-$ denote the reverse ordered partition of $\mathcal{X}$ and a part of $\mathcal{X}^-$ is therefore defined by $\mathcal{X}^-(i) = \mathcal{X}(|\mathcal{X}| - i + 1)$. For a vertex $x \in \mathcal{X}(j)$, we define its **left neighborhood** as $LN(x, \mathcal{X}) = \{y \in N(x) | y \in \mathcal{X}(k) \text{ and } k < j\}$.

During the proof, we need to consider the label of a vertex at some step of the execution of a LBFS. So for an unvisited vertex x, let us denote by $label_y(x)$ as the label of x at the step where y is chosen in the LBFS.

Definition 5.3.1. Let $\mathcal{X}$ be an ordered partition of the vertices of G a cocomparability graph.

We will say that $\mathcal{X}$ is **left-modular** if and only if $\forall j$, $1 \leq j \leq |\mathcal{X}|$, if $x, y \in \mathcal{X}(j)$ then $LN(x, \mathcal{X}) = LN(y, \mathcal{X})$.

Definition 5.3.2. Let $\mathcal{X}$ be an ordered partition of the vertices of G a cocomparability graph.

We will say that $\mathcal{X}$ is **compatible** with an ordering τ if and only if $\forall x \in \mathcal{X}(j)$, $\forall y \in \mathcal{X}(k)$, $j < k$ and $xy \notin E(G)$ then $x <_\tau y$.

Definition 5.3.3. Let $\mathcal{X}$ be an ordered partition of the vertices of G a cocomparability graph.

We will say that $\mathcal{X}$ is a **cocomp ordered partition** if and only if $\mathcal{X}$ is compatible with a cocomp ordering.

5.3.3 The proof

Let us start by proving some propositions about cocomp ordered partitions.

Proposition 5.3.4. *Let $G = (V(G), E(G))$ be a cocomparability graph and $\mathcal{X}$ be a cocomp ordered partition of $V(G)$. If $x, y \in \mathcal{X}(j)$, $xy \notin E(G)$ then $LN(x, \mathcal{X}) \subseteq LN(y, \mathcal{X})$ or $LN(y, \mathcal{X}) \subseteq LN(x, \mathcal{X})$.*

Proof. Assume for contradiction that there exists $x, y \in \mathcal{X}(j)$ such that $xy \notin E(G)$, $LN(x, \mathcal{X}) \nsubseteq LN(y, \mathcal{X})$ and $LN(y, \mathcal{X}) \nsubseteq LN(x, \mathcal{X})$. Since $\mathcal{X}$ is a cocomp ordered partition, let τ be a cocomp ordering such that $\mathcal{X}$ is compatible with τ. We now have two cases: either $y <_\tau x$ or $x <_\tau y$.

Using symmetry, let us only consider the case where $y <_\tau x$. Since $LN(x, \mathcal{X}) \nsubseteq LN(y, \mathcal{X})$, there exists $u \in LN(x, \mathcal{X}) - LN(y, \mathcal{X})$ such that $u \in \mathcal{X}(g)$ with $g < j$. Since $\mathcal{X}$ is compatible with τ and $uy \notin E(G)$, we know that $u <_\tau y$. But now u, y, x is an umbrella, therefore contradicting the fact that τ is a cocomp ordering. □

Proposition 5.3.5. *Let $G = (V(G), E(G))$ be a cocomparability graph, let σ_{i-1} be a LBFS of G and let $\sigma_i = LBFS^+(G, \sigma_{i-1})$. If $\mathcal{X}_{i-1}$ is a cocomp left-modular ordered partition compatible with σ_{i-1}, then $\mathcal{X}^-_{i-1}$ is a cocomp ordered partition compatible with σ_i.*

Proof. Assume for contradiction that $\mathcal{X}^-_{i-1}$ is not compatible with σ_i. So we know that there exists a counter-example i.e. two vertices $x \in \mathcal{X}^-_{i-1}(j)$ and $y \in \mathcal{X}^-_{i-1}(k)$ such that: $j < k$, $xy \notin E(G)$, $y <_{\sigma_{i-1}} x$ and $y <_{\sigma_i} x$ (which means that y is visited before x during $LBFS^+(G, \sigma_{i-1})$ and y appears before x in σ_{i-1}). Let x, y be the leftmost pair of vertices in σ to be a counter-example.

Because of the + rule used in LBFS$^+$ and since y appears before x in σ_{i-1}, at the time y has been chosen by $LBFS^+$, we had $label_y(y) > label_y(x)$. Therefore y has a private neighbour u such that $u <_{\sigma_i} y$. Suppose $u \in \mathcal{X}^-_{i-1}(l)$. We have three cases: $j < l$, $l = j$ and $l < j$. In the first cases, the couple (u, x) contradicts the fact that (y,x) is the leftmost counter-example. In the second case the vertices x, u contradicts the fact that $\mathcal{X}_{i-1}$ is left-modular. In the third case, in any order τ compatible with $\mathcal{X}^-_{i-1}$ we have $u <_\tau x <_\tau y$ which is an umbrella. Therefore it contradicts the fact that the ordered partition $\mathcal{X}_{i-1}$ is a cocomp ordered partition. □

Proposition 5.3.6. *Let $G = (V(G), E(G))$ be a cocomparability graph, let σ be a LBFS of G and $\mathcal{X}$ be a left-modular ordered partition compatible with σ. Then the restriction of σ to a part $\mathcal{X}(i)$ is a legitimate LBFS ordering of $G(\mathcal{X}(i))$.*

Proof. Assume for contradiction that the restriction of σ to a part $\mathcal{X}(i)$ of the partition $\mathcal{X}$ is not an LBFS ordering. Therefore using theorem 1.3.4 on σ, we deduce that there exists three vertices a, b, $c \in \mathcal{X}(i)$ such that $ac \in E(G)$, $ab \notin E(G)$, $a <_\sigma b <_\sigma c$ and $\nexists e \in \mathcal{X}(i)$ such that $eb \in E(G)$, $ec \notin E(G)$ and $e <_\sigma a$. Since σ is a LBFS ordering, there exists d such that $db \in E(G)$, $dc \notin E(G)$ and $d <_\sigma a$. Since the partition is left-modular, d must belong to a part $\mathcal{X}(j)$ of $\mathcal{X}$ and $i \leq j$. Since $\nexists e \in \mathcal{X}(i)$ such that $eb \in E(G)$, $ec \notin E(G)$ and $e <_\sigma a$ we further know that $i < j$. But now $d <_\sigma c$ contradicts the fact that $\mathcal{X}$ is compatible with σ. □

Let us now present our auxiliary algorithm (Algorithm 10) called Partition, which computes from a partition $\mathcal{X}_{i-1}$ and an ordering σ_i of the vertices a new partition denoted by $\mathcal{X}_i = Partition(\mathcal{X}_{i-1}, \sigma_i)$.

This algorithm can be seen as a partition refinement algorithm. See Figure 5.2 for a drawing of the behaviour of the algorithm.

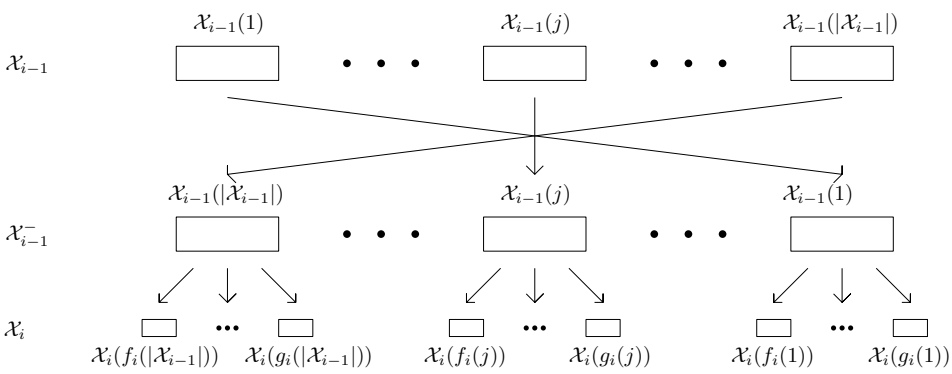


Figure 5.2: Partition algorithm

Let us illustrate this algorithm on the example of Figure 5.1 and the series of orderings $\sigma_1 =< d, c, b, f, a, e >$, $\sigma_2 =< e, f, b, d, c\ a >$, $\sigma_3 =< a, c, b, d, e, f >$, $\sigma_4 =< f, e, d, b, c, a >$, $\sigma_5 =< a, c, b, d, e, f >$, $\sigma_6 =< f, e, d, b, c, a >$. By convention we set $\mathcal{X}_1 =< V >$. In this example we have $\mathcal{X}_1 =< \{a, b, c, d, e, f\} >$. To build $\mathcal{X}_2$, the input is σ_2, $\mathcal{X}_1$. So for the first part of $\mathcal{X}_2$ we put the leftmost vertex

Algorithm 10: Partition

Data: A vertex ordering σ_i and a partition $\mathcal{X}_{i-1}$ of $V(G)$

Result: An ordered partition $\mathcal{X}_i = Partition(\mathcal{X}_{i-1}, \sigma_i)$

```
1  X_i ← (); % { Initialization of the new partition } %
2  l ← 1; % { X_i(l) denotes the part of X_i that we are building } %
3  for (j ← |X_{i-1}| downto 1) do
4      T_1^{(i,j)} ← X_{i-1}(j); % { T_1^{(i,j)} is the set of vertices not already puts in X_i
       } %
5      f_i(j) ← l % { index of the first part using the vertices of X_{i-1}(j)} %;
6      k ← 1;
7      while (T_k^{(i,j)} ≠ ∅) do
8          Let x be the leftmost vertex of T_k^{(i,j)} in σ_i;
9          if (X_{i-1}(j) = T_k^{(i,j)}) then
10             X_i(l) ← {x};
11         else
12             X_i(l) ← {u ∈ T_k^{(i,j)} | LN(x, X_i) = LN(u, X_i)} % { Put in the new
               part all the vertices with the same left neigborhood than x } %;
13         end
14         T_{k+1}^{(i,j)} ← T_k^{(i,j)} − X_i(l);
15         k ← k + 1;
16         l ← l + 1;
17     end
18     g_i(j) ← l % { index of the last part using the vertices of X_{i-1}(j)} %;
19     j ← j + 1;
20 end
```

in σ_2 which is e. In the second part we will put f and all the vertices withe same neighbourhood, therefore the second part will be equal to $\{f, b\}$. And we continue like that. At the end we find $\mathcal{X}_2 =< \{e\}, \{f, b\}, \{d\}, \{c\}, \{a\} >$. For $\mathcal{X}_3$ the algorithm will output the partition $< \{a\}, \{c\}, \{d\}, \{b\}, \{f\}, \{e\} >$. It must be noticed that if the partition $\mathcal{X}_{i-1}$ is the trivial partition (made up with singletons), then the algorithm outputs $\mathcal{X}_i = \mathcal{X}_{i-1}^{-}$. So the algorithm will output the partition $\mathcal{X}_4 =< \{e\}, \{f\}, \{b\}, \{d\}, \{c\}, \{a\} >$, $\mathcal{X}_5 =< \{a\}, \{c\}, \{d\}, \{b\}, \{f\}, \{e\} >= \mathcal{X}_3$

and $\mathcal{X}_6 =< \{e\}, \{f\}, \{b\}, \{d\}, \{c\}, \{a\} >= \mathcal{X}_4$.

It should be noticed that σ_3 and $\mathcal{X}_3$ (resp. σ_4 and $\mathcal{X}_4$) do not provide the same ordering of the vertices of G, but σ_3 and $\mathcal{X}_3$ (resp. σ_4 and $\mathcal{X}_4$) are not only both cocomp orderings but also linear extensions of the same transitive orientation of $\overline{G}$.

First two easy observations :

Proposition 5.3.7. *For every partition $\mathcal{X}_{i-1}$ and every ordering σ_i, $\mathcal{X}_i = Partition(\mathcal{X}_{i-1}, \sigma_i)$ is left-modular.*

Proof. Each time a part $\mathcal{X}_i(j)$ of $\mathcal{X}_i$ is built, we only put together in $\mathcal{X}_i(j)$ the set of vertices $\{u \in T_k^{(j)} | LN(x, \mathcal{X}_{i-1}) = LN(u, \mathcal{X}_{i-1})\}$ in it. Therefore the new partition $\mathcal{X}_i$ is left-modular. □

Proposition 5.3.8. *For every partition $\mathcal{X}_{i-1}$ and every ordering σ_i, $\mathcal{X}_i = Partition(\mathcal{X}_{i-1}, \sigma_i)$ is an ordered refinement of $\mathcal{X}_{i-1}$.*

Proof. A part of $\mathcal{X}_{i-1}$ can only be refined during the Partition algorithm. Furthermore if $\mathcal{X}_{i-1}(i)$ and $\mathcal{X}_{i-1}(j)$ are two parts of $\mathcal{X}_{i-1}$, with $i < j$, then for every $\mathcal{X}_i(p) \subseteq \mathcal{X}_{i-1}(i)$ and $\mathcal{X}_i(q) \subseteq \mathcal{X}_{i-1}(j)$, we necessarily have: $q < p$.

□

Therefore the above Partition algorithm can be seen as a non-standard partition refinement algorithm see [HMPV00]. The next theorem establishes the relationships between the algorithms Repeated LBFS$^+$ and Partition. Roughly speaking we will prove that the Partition algorithm maintains the following invariant:

There exists a cocomp ordering compatible with $\mathcal{X}$.

Furthermore Repeated LBFS$^+$ and Partition agree on non edges, and at the end the results obtained by the two algorithms are two cocomp orderings which are two linear extensions of the same transitive orientation of $\overline{G}$.

Before going any further, let us fix some notations to capture the behavior of the Partition algorithm. The notations are illustrated in Figure 5.3. We define a sequence of partitions:

$\mathcal{P}_0^{(i,j)} = \mathcal{X}_{i-1}^-$ and

$\mathcal{P}_k^{(i,j)} =< \mathcal{X}_{i-1}(|\mathcal{X}_{i-1}|),\ldots,\mathcal{X}_{i-1}(j+1), \mathcal{X}_i(f_i(j)),\ldots \mathcal{X}_i(f_i(j))+k-1), T_{k+1}^{(i,j)}, \mathcal{X}_{i-1}(j-1), ..., \mathcal{X}_{i-1}(1) >$ for $1 \leq j \leq |\mathcal{X}_{i-1}|$, $1 \leq k \leq g_i(j) - f_i(j)$.

This series of partitions $\mathcal{P}_k^{(i,j)}$ describes how the part $\mathcal{X}_{i-1}(j)$ is divided into parts of the partition $\mathcal{X}_i$, as described in Figure 5.3. $\mathcal{P}_k^{(i,j)}$ is exactly the current partition of $\mathcal{X}_i$ at step 18 of the Partition algorithm.

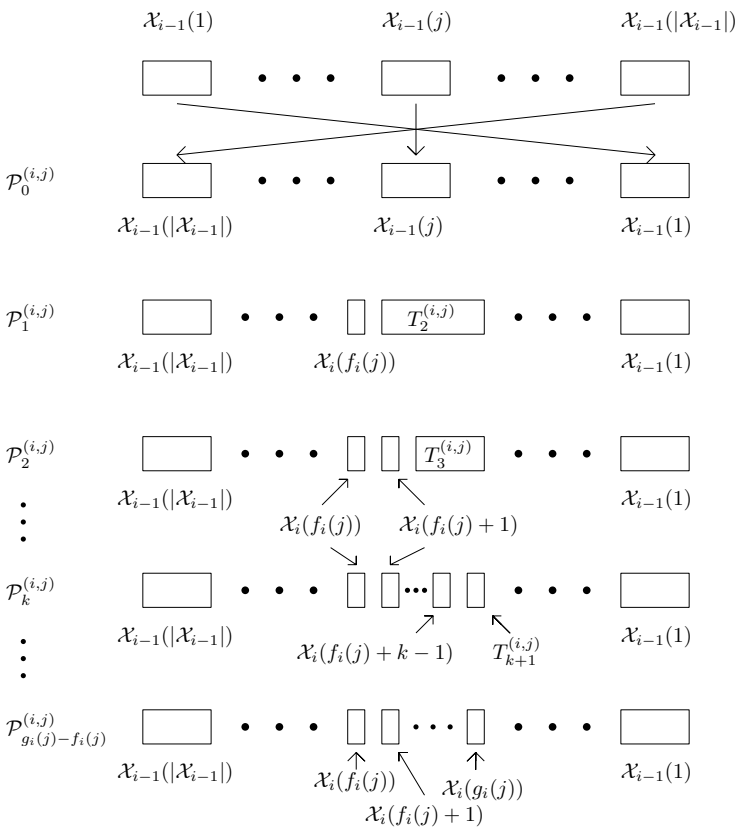


Figure 5.3: Illustration of the notation defined for the Partition algorithm

Theorem 5.3.9. *Let G be a cocomparability graph and let $\sigma_1, ..., \sigma_n$ be the series of LBFS orderings produced by Repeated LBFS$^+$ and $\mathcal{X}_1, ..., \mathcal{X}_n$ be the series of ordered partitions such that $\mathcal{X}_1 =< V(G) >$ and $\mathcal{X}_i = Partition(\mathcal{X}_{i-1}, \sigma_i)$ for $1 < i \leq n$. Then for every i, $1 \leq i \leq n$, $\mathcal{X}_i$ is a left-modular cocomp ordered partition compatible with σ_i.*

Proof. In proposition 5.3.7, we have already proved that $\mathcal{X}_i$ is left-modular and so we only have to prove that $\mathcal{X}_i$ is a cocomp ordered partition compatible with σ_i. We define $\mathcal{X}_0 =< V(G) >$. The proof is by induction on i. As induction hypothesis we use the main common invariants of the two algorithms :

(1) $\mathcal{X}_i$ is a cocomp ordered partition compatible with σ_i.

(2) for all $x, y \in \mathcal{X}_{i-1}(j)$, $x \in \mathcal{X}_i(p)$, $y \in \mathcal{X}_i(q)$, $p < q$ then $x <_{\sigma_i} y$.

For the initial case, since $\mathcal{X}_1 =< V(G) >$, it is clear that $\mathcal{X}_1$ is compatible with σ_1. Since G is a cocomparability graph, there exists a cocomp ordering τ and since $\mathcal{X}_1 =< V(G) >$, $\mathcal{X}_1$ is compatible with τ. So $\mathcal{X}_1$ is a cocomp ordered partition. For **(2)**, since we define $\mathcal{X}_0 =< V(G) >$ and $\mathcal{X}_1 =< V(G) >$, **(2)** is trivially true at the first step. Therefore the induction hypothesis **(1)** and **(2)** are true for $i = 1$.

For the induction step, suppose that the hypotheses **(1)** and **(2)** are true for every l such that $1 \leq l < i$. In order to prove that the hypotheses **(1)** and **(2)** are true for i, let us focus our attention on one part $\mathcal{X}_i(j)$ and study the behavior of the Partition algorithm on this part. We again use induction and the hypotheses are:

(i) $\mathcal{P}_k^{(i,j)}$ is a cocomp ordered partition for $1 \leq k \leq g_i(j) - f_i(j)$.

(ii) for all $x, y \in \mathcal{X}_{i-1}(j)$, $x \in \mathcal{P}_k^{(i,j)}(p)$, $y \in \mathcal{P}_k^{(i,j)}(q)$, with $p < q$ then $x <_{\sigma_i} y$.

For the initial case, since we put in the first part only the leftmost vertex of $T_1^{(i,j)}$, the induction hypothesis **(ii)** is true for $k = 1$. In the next claim, we prove that the second hypohesis is also true.

Claim 5.3.10. *If the induction hypothesis **(1)** is true for $i - 1$, then condition **(i)** is true for k=1.*

Proof. Let y be the leftmost vertex of $T_1^{(i,j)}$, with respect to σ_i. Let $S_y = \{x \in T_1^{(i,j)} | LN(x, \mathcal{P}_0^{(i,j)}) = LN(y, \mathcal{P}_0^{(i,j)})\}$. We will first show that there exists a cocomp ordering τ compatible with $\mathcal{X}_{i-1}^- = \mathcal{P}_0^{(i,j)}$ such that for all $u \in S_y$, for all $v \in T_1^{(i,j)} - S_y$, $uv \notin E(G)$, $u <_\tau v$. Then we will prove that $\mathcal{P}_1^{(i,j)}$ is compatible with some cocomp ordering.

Assume for contradiction that there is no cocomp ordering τ compatible with $\mathcal{X}_{i-1}^- = \mathcal{P}_0^{(i,j)}$ such that for all $u \in S_y$, for all $v \in T_1^{(i,j)} - S_y$, $uv \notin E(G)$, $u <_\tau v$. So in any cocomp ordering γ such that $\mathcal{P}_0^{(i,j)}$ is compatible with γ, $\exists u \in S_y$, $\exists v \in T_1^{(i,j)} - S_y$, $uv \notin E(G)$, $v <_\tau u$. Since $\mathcal{P}_0^{(i,j)} = \mathcal{X}_{i-1}^-$, by induction hypothesis **(1)** we know that $\mathcal{P}_0^{(i,j)}$ is cocomp and compatible with σ_i. So using proposition 5.3.4 we know that either $LN(u, \mathcal{P}_0^{(i,j)}) \subseteq LN(v, \mathcal{P}_0^{(i,j)})$ or $LN(v, \mathcal{P}_0^{(i,j)}) \subseteq LN(u, \mathcal{P}_0^{(i,j)})$.

In the first case, since $u, y \in S_y$ we know that $LN(y, \mathcal{P}_0^{(i,j)}) = LN(u, \mathcal{P}_0^{(i,j)})$ and so we deduce that $LN(y, \mathcal{P}_0^{(i,j)}) \subseteq LN(v, \mathcal{P}_0^{(i,j)})$. Since $v \notin S_y$, we further know that $LN(y, \mathcal{P}_0^{(i,j)}) \subsetneq LN(v, \mathcal{P}_0^{(i,j)})$. Using theorem 5.3.5 we know that $\mathcal{P}_0^{(i,j)} = \mathcal{X}_{i-1}^-$ is compatible with σ_i. Therefore for all $z \in LN(v, \mathcal{P}_0^{(i,j)}) - LN(y, \mathcal{P}_0^{(i,j)})$, $z <_{\sigma_i} y$. But now at the time y has been chosen by LBFS in σ_i, since $LN(y, \mathcal{P}_0^{(i,j)}) \subsetneq LN(v, \mathcal{P}_0^{(i,j)})$ we would have $label_y(u) > label_y(y)$. Therefore it yields $u <_{\sigma_i} y$ which contradicts the fact that y is the leftmost vertex of $T_0^{(i,j)}$ with respect to σ_i.

In the second case, let $z \in LN(u, \mathcal{P}_0^{(i,j)}) - LN(v, \mathcal{P}_0^{(i,j)})$. Since $\mathcal{P}_0^{(i,j)}$ is compatible with γ, we know that $z <_\gamma v$. But now z, v, u form an umbrella, which contradicts the fact that γ is a cocomp ordering. Therefore there must exists a cocomp ordering τ such that for all $u \in S_y$, for all $v \in T_1^{(i,j)} - S_y$, $uv \notin E(G)$, $u <_\tau v$.

We now show that there exists a cocomp ordering τ such that for all $v \in \mathcal{X}_i(j)$, $y <_\tau v$ which implies that $\mathcal{P}_1^{(i,j)}$ is a cocomp ordered partition compatible with σ_i. Since there exists a cocomp ordering τ such that for all $u \in S_y$, for all $v \in T_1^{(i,j)} - S_y$, $uv \notin E(G)$, $u <_\tau v$, we have to show that there exists a cocomp ordering for the graph induced by the vertices of S_y such that y is an extremity of it. Since $\mathcal{X}_{i-1}$ is cocomp, left-modular and compatible with σ_{i-1}, using theorem 5.3.6 on $\mathcal{X}_{i-1}$, we know that the restriction of σ_{i-1} to $\mathcal{X}_{i-1}(j)$ is an LBFS ordering for the graph induced by the vertices of $\mathcal{X}_{i-1}$. Since at the time y has been chosen in σ_i all the vertices of S_y had the same label, by the + rule we deduce that y was the last of the vertices of S_y. Therefore using theorem 5.2.2, we deduce that for the subgraph induced by the set of vertices $\{z | z \in \mathcal{X}_{i-1}(j) \text{ and } z <_{\sigma_{i-1}} y\}$, there exists a cocomp ordering in which y is last. So for the graph induced by S_y, there exists a cocomp ordering in which y is last. $\square$

So far, we proved the starting case. Let us now prove the inductive step with

the two next claims.

Claim 5.3.11. *If the induction hypothesis **(1)** is true for $i-1$ and the hypothesis **(i), (ii)** are true for $k-1$, then the hypotheses **(i)** is true for k.*

Proof. Let y be the leftmost vertex of $T_k^{(i,j)} - T_{k+1}^{(i,j)}$ with respect to σ_i. Since $\mathcal{P}_{k-1}^{(i,j)}$ is a cocomp ordered partition, let τ be a cocomp ordering such that $\mathcal{P}_{k-1}^{(i,j)}$ is compatible with τ. Assume for contradiction that $\mathcal{P}_k^{(i,j)}$ is not a cocomp ordered partition. Therefore $\mathcal{P}_k^{(i,j)}$ is not compatible with τ and there exists $v \in T_{k+1}^{(i,j)}$, $u \in T_k^{(i,j)} - T_{k+1}^{(i,j)}$, $uv \notin E(G)$, $v <_\tau u$. Since $\mathcal{P}_g^{(i,j)}$ is a cocomp ordered partition, using claim 5.3.4 on v, u we know that either $LN(u, \mathcal{P}_{k-1}^{(i,j)}) \subseteq LN(v, \mathcal{P}_{k-1}^{(i,j)})\}$ or $LN(v, \mathcal{P}_{k-1}^{(i,j)}) \subsetneq LN(u, \mathcal{P}_{k-1}^{(i,j)})\}$.

In the first case, since $u, y \in T_k^{(i,j)} - T_{k+1}^{(i,j)}$, we know that $LN(y, \mathcal{P}_{k-1}^{(i,j)}) = LN(u, \mathcal{P}_{k-1}^{(i,j)})$ and so we deduce that $LN(y, \mathcal{P}_{k-1}^{(i,j)}) \subseteq LN(v, \mathcal{P}_{k-1}^{(i,j)})$. Since $v \notin T_k^{(i,j)} - T_{k+1}^{(i,j)}$, we further know that $LN(y, \mathcal{P}_{k-1}^{(i,j)}) \subsetneq LN(v, \mathcal{P}_{k-1}^{(i,j)})$. Since the induction hypothesis **(1)** is true for $i-1$ and hypothesis **(ii)** is true for $k-1$, we know that $\mathcal{P}_{k-1}^{(i,j)}$ is compatible with σ_i. Therefore for all $z \in LN(v, \mathcal{P}_{k-1}^{(i,j)}) - LN(y, \mathcal{P}_{k-1}^{(i,j)})$, $z <_{\sigma_i} y$. But at the time y has been chosen by LBFS, we would have $label_y(v) > label_y(y)$. Therefore it yields $v <_{\sigma_i} y$ which contradicts the fact that y is the leftmost vertex of $T_k^{(i,j)} - T_{k+1}^{(i,j)}$ with respect to σ_i.

In the second case, let $z \in LN(u, \mathcal{P}_{k-1}^{(i,j)}) - LN(v, \mathcal{P}_{k-1}^{(i,j)})$. Since τ is compatible with $\mathcal{P}_{k-1}^{(i,j)}$, we know that $z <_\tau v$. But now z, v, u forms an umbrella, which contradicts the fact that τ is a cocomp ordering. □

In the next claim, we show that the second hypothesis is also true.

Claim 5.3.12. *If the induction hypotheses **(1), (2)** are true for $i-1$ and the induction hypothesis **(i), (ii)** are true for $k-1$, then hypotheses **(ii)** is true for k.*

Proof. Assume for contradiction that there exists $u \in T_k^{(i,j)} - T_{k+1}^{(i,j)}$, $v \in T_{k+1}^{(i,j)}$ such that $v <_{\sigma_i} u$. Let y be the leftmost vertex of $T_k^{(i,j)} - T_{k+1}^{(i,j)}$ with respect to σ_i. Since v is not in the current part, we have two cases: either $LN(v, \mathcal{P}_{k-1}^{(i,j)}) - LN(u, \mathcal{P}_{k-1}^{(i,j)}) \neq \emptyset$ or $LN(v, \mathcal{P}_{k-1}^{(i,j)}) \subsetneq LN(u, \mathcal{P}_{k-1}^{(i,j)})$.

In the first case, by the induction hypothesis we know that $\mathcal{P}_{k-1}^{(i,j)}$ is compatible with σ_i, and so we deduce that $\forall x \in LN(v, \mathcal{P}_{k-1}^{(i,j)}) - LN(y, \mathcal{P}_{k-1}^{(i,j)})$, $x <_{\sigma_i} y$. So at the time we visit y, if we choose y, this implies that $label_y(y) > label_y(v)$. Since

$LN(y, \mathcal{P}_{k-1}^{(i,j)}) = LN(u, \mathcal{P}_{k-1}^{(i,j)})$, we must also have $label_y(u) > label_y(v)$. Therefore the LBFS must visit u before v in σ_i which contradicts our choice of u, v.

In the second case, suppose that there exists $w \in LN(u, \mathcal{P}_{k-1}^{(i,j)}) - LN(v, \mathcal{P}_{k-1}^{(i,j)})$ such that $w <_{\sigma_i} y$. This implies that $label_y(y) > label_y(v)$ and so $label_y(u) > label_y(v)$. Therefore the LBFS must visit u before v in σ_i which contradicts our choice of u, v. Therefore assume for contradiction that $\nexists w \in LN(u, \mathcal{P}_{k-1}^{(i,j)}) - LN(v, \mathcal{P}_{k-1}^{(i,j)})$ such that $w <_{\sigma_i} y$. Let $x \in LN(u, \mathcal{P}_{k-1}^{(i,j)}) - LN(v, \mathcal{P}_{k-1}^{(i,j)})$. Therefore $y <_{\sigma_i} x$. We have again two cases: either $x \in \mathcal{X}_{i-1}(j)$ or $x \in \mathcal{X}_{i-1}(g)$ with $j < g$. In the first case, using the induction hypothesis **(ii)** we deduce that $x <_{\sigma_i} y$ which contradicts the assumption that $y <_{\sigma_i} x$. In the second case, since $\mathcal{X}_{i-2}$ is left-modular, x, u, v, y were in the same part in $\mathcal{X}_{i-2}$. Therefore x has been separated from u, v, y in $\mathcal{X}_{i-1}$. But now using induction hypothesis **(2)** we deduce that $u, v, y <_{\sigma_{i-1}} x$. So in σ_i, since y has been chosen by the $LBFS^+$ before x, we deduce that $label_y(y) > label_y(x)$. Let $z \in N(y) - N(x)$ be such that $z <_{\sigma_i} y$. Since $\nexists w \in LN(u, \mathcal{P}_{k-1}^{(i,j)}) - LN(v, \mathcal{P}_{k-1}^{(i,j)})$ such that $w <_{\sigma_i} y$, we know that $wv \in E(G)$ and since $LN(u, \mathcal{P}_{k-1}^{(i,j)}) = LN(y, \mathcal{P}_{k-1}^{(i,j)})$ we also have $uz \in E(G)$. Let $z \in \mathcal{X}_{i-1}(h)$. We now have three cases: $g < h$; $g = h$; $h < g$. In the first case $g < h$, v, x, z contradict the fact that $\mathcal{X}_{i-1}$ is cocomp. In the second case, x, z contradict the fact that $\mathcal{X}_{i-1}$ is left-modular. In the third case, since $\mathcal{X}_{i-1}^-$ is compatible with σ_i we know that $x <_{\sigma_i} w$ which contradicts our choice of w. Thereby there must exist $w \in LN(u, \mathcal{P}_{k-1}^{(i,j)}) - LN(v, \mathcal{P}_{k-1}^{(i,j)})$ such that $w <_{\sigma_i} y$ and so LBFS must visit u before v in σ_i which contradicts our choice of u, v. □

We now show that for i, hypotheses **(1)** and **(2)** are true.

Let first take care of **(2)**. Since **(ii)** is true for any part of $\mathcal{X}_{i-1}$, hypothesis **(2)** is true for i.

Let us now take care of **(1)**. First let us show that $\mathcal{X}_i$ is compatible with σ_i, then that $\mathcal{X}_i$ is a cocomp ordered partition.

We prove that $\mathcal{X}_i$ is compatible with σ_i. Let us consider two vertices x, y such that $x \in \mathcal{X}_i(p)$, $y \in \mathcal{X}_i(q)$, $xy \notin E(G)$, $p < q$. Now we have two cases: either x, y were in different parts in $\mathcal{X}_{i-1}$ or they were in the same part in $\mathcal{X}_{i-1}$. In the first case, since by induction we assume $\mathcal{X}_{i-1}$ is cocomp, left-modular and compatible with σ_{i-1}, using theorem 5.3.5 on $\mathcal{X}_{i-1}$ and σ_{i-1} we know that $\mathcal{X}_{i-1}^-$ is cocomp and compatible with σ_i. Therefore $x <_{\sigma_i} y$. In the second case, since condition **(ii)**

is true for any part $\mathcal{X}_{i-1}(j)$, we deduce that $x <_{\sigma_i} y$. therefore $\mathcal{X}_i$ is compatible with σ_i.

We now prove that $\mathcal{X}_i$ is a cocomp ordered partition. Since **(i)** is true for any part of $\mathcal{X}_{i-1}$, we know that $\mathcal{P}^{(i,j)}_{g_i(j)-f_i(j)}$ is a cocomp ordered partition. Let us consider $\tau_1, \ldots, \tau_{|\mathcal{X}_{i-1}|}$ to be a series of orderings such that for $1 \leq j \leq |\mathcal{X}_{i-1}|$, τ_j is a cocomp ordering compatible with $\mathcal{P}^{(i,j)}_{g_i(j)-f_i(j)}$. Let $\tau_j[\mathcal{X}_{i-1}(j)]$ be the ordering of the vertices of $\mathcal{X}_{i-1}(j)$ induced by τ_j. Let $\gamma = \tau_1[\mathcal{X}_{i-1}(1)]$,..., $\tau_j[\mathcal{X}_{i-1}(j)]$,..., $\tau_{|\mathcal{X}_{i-1}|}[\mathcal{X}_{i-1}(|\mathcal{X}_{i-1}|)]$. We now show that γ is a cocomp ordering. Assume for contradiction that γ is not a cocomp ordering. Therefore there exists an umbrella $x,\ y,\ z$ with $x <_\gamma y <_\gamma z$. We now have a couple of cases to handle: x belongs to a part and $y,\ z$ to another one; $x,\ y$ belongs to a part and z to another one; $x,\ y,\ z$ belong to three different parts of $\mathcal{X}_{i-1}$; $x,\ y,\ z$ belong to the same part of $\mathcal{X}_{i-1}$; In the first case, let $y,\ z \in \mathcal{X}_{i-1}(b)$, $x \in \mathcal{X}_{i-1}(a)$ and $a < b$. In τ_b, we see that we must have $x <_{\tau_b} y <_{\tau_b} z$ which contradicts τ_b being a cocomp ordering. The same argument as in the first case works for the second case. In the third case, we have a contradiction to $\mathcal{X}_{i-1}$ being a cocomp ordered partition. In the last case, let $x,\ y,\ z \in \mathcal{X}_{i-1}(j)$. We now contradict τ_j being a cocomp ordering. Therefore γ is a cocomp ordering and $\mathcal{X}_i$ is a cocomp ordered partition.

□

With the last theorem we can now prove that Repeated LBFS$^+$ algorithm computes a cocomp ordering

Theorem 5.3.13. *$G = (V(G), E(G))$ is a cocomparability graph if and only if the Repeated LBFS$^+$ algorithm computes a cocomp ordering in at most $|V(G)|$ LBFS$^+$.*

Proof. Suppose that Repeated LBFS$^+$ algorithm computes a cocomp ordering in at most $|V(G)|$ LBFS$^+$. Hence G is a cocomparability graph.

Conversely, suppose G a cocomparability graph. First, we will prove that in $\mathcal{X}_n$, if each part contains only one vertex, then that σ_n is a cocomp ordering. We do the proof by induction and as an invariant we show that at step i, $1 \leq i \leq n$ there exists at least i non empty parts in the partition $\mathcal{X}_i$.

For the initial step, since $\mathcal{X}_1 =< V(G) >$ the induction hypothesis is true.

For the inductive step, since for each part $\mathcal{X}_{i-1}(j)$ we create in $\mathcal{X}_i$ a part that contains the leftmost vertex of $\mathcal{X}_i$ with respect to σ_i, moreover every part that

contains at least two vertices is divided. Therefore if there is a part that contains two vertices then the induction hypothesis is true. If every part contains only one vertex in $\mathcal{X}_{i-1}$, we deduce that we have n parts in $\mathcal{X}_{i-1}$. Thereby, since $i \leq n$, the induction hypothesis is also true.

Assume for contradiction that σ_n is not a cocomp ordering. Thus there exists an umbrella $u,\ v,\ w$, such that $u <_{\sigma_n} v <_{\sigma_n} w$. Since $\mathcal{X}_n$ is composed of n parts, we know that $u,\ v,\ w$ are in different parts. Let $u \in \mathcal{X}_n(a)$, $v \in \mathcal{X}_n(b)$ and $w \in \mathcal{X}_n(c)$. Since using theorem 5.3.9 on $\mathcal{X}_n$, σ_n, we deduce that $\mathcal{X}_n$ is compatible with σ_n and so we further know that $a < b < c$. But now in every cocomp ordering τ such that $\mathcal{X}_n$ is compatible with τ we must have $u <_\tau v <_\tau w$. But $u,\ v,\ w$, is an umbrella which contradicts τ to be a cocomp ordering. Thus there exists no cocomp ordering τ such that $\mathcal{X}_n$ is compatible with τ, and $\mathcal{X}_n$ is not a cocomp ordered partition. But this contradicts theorem 5.3.9. □

Unfortunately, it could be the case that at each step $\mathcal{X}_i$ is divided into 2 parts, with one containing few vertices. Therefore the worst time complexity cannot be improved (see also [Ma]). We will now turn our attention to look at the behavior of Repeated LBFS$^+$. We start by recalling one well known characterization of chordal graphs and of well known theorem about LBFS in chordal graph.

Theorem 5.3.14. *[RTL76] A graph $G = (V(G), E(G))$ is a chordal graph if and only if and only if there exists an ordering τ of its vertices such that for every triple of vertices a, b, c such that $a <_\tau b <_\tau c$,*

$$ac, bc \in E(G) \implies ab \in E(G). \tag{5.1}$$

Such an ordering is called a perfect elimination ordering.

Theorem 5.3.15. *[RTL76] A graph G is a chordal graph if and only if LBFS produces a perfect elimination ordering.*

Now it is now easy to prove the following one:

Theorem 5.3.16. *G is an interval graph if and only if Repeated LBFS$^+$ algorithm computes an interval ordering in at most $|V(G)|$ LBFS$^+$.*

Proof. Let G be an interval graph. Using theorem 1.4.8 on G, we deduce that G is a chordal graph and a cocomparability graph. Since G is a chordal graph and σ_n is a LBFS ordering of the vertices of G, we deduce using theorem 5.3.15 that σ_n is a perfect elimination ordering. Using the previous theorem, we know that σ_n is also a cocomp ordering.

Let now show that σ_n is an interval ordering. Let us consider a, b, c such that $ac \in E(G)$ and $a <_{\sigma_n} b <_{\sigma_n} c$. Since σ_n is a cocomp ordering we know that $ab \in E(G)$ or $bc \in E(G)$. In the case where $bc \in E(G)$, since σ_n is a perfect eliminitation ordering we know that $ab \in E(G)$. Therefore $ac \in E(G)$ implies that $ab \in E(G)$. So σ_n is an interval ordering.

Conversely, if Repeated LBFS$^+$ algorithm computes an interval ordering in at most $|V(G)|$ LBFS$^+$ then G is an interval graph. □

Theorem 5.3.17. *Repeated LBFS$^+$ has complexity $O(nm)$.*

Proof. The description of a linear time implementation of LBFS can be found in [RTL76, HMPV00]. A linear time implementation of LBFS$^+$ has been explained in [COS09, Cor04b]. Therefore the Repeated LBFS$^+$ can be implemented in $O(nm)$. □

Since it is well-known (see for example [HMPV00]) that given G and using partition refinement a LBFS can be computed on $\overline{G}$ in linear time in the size of G, this immediately yields a very simple algorithm easy to implement that computes in $O(nm)$ a transitive orientation of a comparability graph.

Corollary 5.3.18. *G is a comparability graph if and only if Repeated LBFS$^+$ algorithm applied on $\overline{G}$ provides a linear extension of a transitive orientation of G in at most $|V(G)|$ LBFS$^+$.*

5.4 Conclusions and perspectives for further work

In this chapter we have presented and proved an algorithm based on a series of LBFS$^+$ to find a cocomp ordering if G is a cocomparability graph. This answers a question asked in [COS09] about the existence of a multisweep LBFS algorithm to find a cocomp ordering. As a byproduct this algorithm also works for interval

graphs and therefore we proved a result claimed in [Ma, CK] but never published. Although the worst case can be achieved, we think that in a similar way that for interval graphs in [COS09], using an extra variant of LBFS the following can be true.

Conjecture 5.4.1. *If G is a cocomparability graph then a series of $O(1)$ LBFS$^+$ followed by a special LBFS always produces a cocomp ordering in $O(n+m)$.*

During our experimentation of Repeated LBFS$^+$ algorithm, for many graphs the series of ordering falls into a loop of size 2, mostly after few steps. Therefore a natural question is :

For an infinite series of LBFS$^+$, what is the size of the final loop ? We prove that after at most n steps this series reaches the set of cocomp orderings and using corollary 1.5.6 the series stays in cocomp orderings. Since the number of cocomp orderings of a given graph is finite, this series necessarily fall into a finite cycle.

Conjecture 5.4.2. *This loop is always of size 2.*

In other words the series of orderings of the Repeated LBFS$^+$ algorithm always reaches a cocomp ordering after which the series alternates between two cocomp orderings.

Algorithm 11: Transitive orientation Algorithm

Data: $G = (V(G), E(G))$ a connected graph

Result: a cocomp ordering of G if and only if G is a cocomparability graph

```
σ_1 ← LBFS(Ḡ);
σ_2 ← LBFS+(Ḡ,σ_1);
σ_3 ← LBFS+(Ḡ,σ_2);
i ← 3;
while σ_i ≠ σ_{i-2} do
    i ← i + 1 ;
    σ_i ← LBFS+(Ḡ,σ_{i-1});
end
Output σ_i;
```

If conjecture 5.4.2 is true, the above algorithm always terminates if the input is a comparability graph in $O(nm)$, and up to our experimentations it seems to be linear in average and therefore very practical.

6

TBLS: a new model for graph search

6.1 Introduction

Some recent applications of graph searches involve a controlled tie-break mechanism in a series of consecutive graph searches, see [Cor04b, COS09, CDH13, DH]. Examples can be found in this thesis but also include the strongly connected components computation using double-DFS [Sha81] and the series of an arbitrary LBFS followed by two LBFS$^+$s used to recognize unit interval graphs [Cor04b]. This motivates a general study of these graph searches equipped with a tie-break mechanism that incorporates such multi-sweep usage of graph searches. This is the goal of the present chapter: to define a framework powerful enough to capture many graph searches either used individually or in a multi-sweep fashion and simple enough to allow general theorems on graph searches. Building on the General Label Search (GLS) framework from [KSB11] we not only simplify their model but also unify their model with the "pattern-conditions" formalism of [CK08]. The framework of this chapter is more restricted than the one presented in chapter 1 but is interesting since most of the classical graph searches fit in it and it allow us to exhibit an algebraic structure.

This chapter is organized as follows. Section 6.2 introduces the Tie-Breaking Label Search (TBLS) formalism to address graph searches. We then illustrate the TBLS by expressing some classical graph searches in this formalism. We will also show the relationship between our formalism and the "pattern-conditions" of search orderings introduced in [CK08] thereby yielding some new pattern-

conditions for various classical searches. In Section 6.4 we introduce two new graph searches. In Section 6.5 we look at the relation between various classical graph searches as well as the two new searches. In Section 6.6 we show that the TBLS and GLS models capture the same set of graph searches. In section 6.7 we propose a unified method for testing, and certifying, if a given ordering of the vertices may be produced by a given graph search; we also present a common implementation framework. Finally, we present a new graph parameter based on our framework.

Many of the results of this chapter is a common work with D.G. Corneil, M. Habib, A. Mamcarz, F. de Montgolfier and submitted [CDH$^+$].

6.2 TBLS, a Tie-Breaking Label Search

A *graph search* is an iterative process that chooses at each step a vertex of the graph and numbers it (from 1 to n). Each vertex is chosen (also said *visited*) exactly once (even if the graph is disconnected). Let us now define a *General Tie-Breaking Label Search* (TBLS). It uses *labels* to decide the next vertex to be visited; $label(v)$ is a subset of $\{1, ..., n\}$. A TBLS is defined on:

1. A graph $G = (V(G), E(G))$ on which the search is performed;
2. A strict partial order $<$ over the label-set $P(\mathbb{N}^+)$;
3. An ordering τ of the vertices of $V(G)$ called the *tie-break permutation.*

The output of $TBLS(G, <, \tau)$ is a permutation σ of $V(G)$, called a $TBLS-$ *ordering* or also the *search ordering* or *visiting ordering.* Let us say a vertex v is *unnumbered* until $\sigma(i) \leftarrow v$ is performed, and then i is its *visiting date.* $label(v)$ is always, thanks to the following algorithm, the set of visiting dates of the neighbors of v visited before v. More specifically $label_i(v)$ for a vertex v denotes the label of v at step i. This formalism identifies a search with the orderings it may produce, as in [CK08], while extending the formalism of General Label Search (GLS) of [KSB11] by the introduction of a *tie-break* ordering τ, making the result of a search algorithm purely deterministic (no arbitrary decision is taken).

Algorithm 12: TBLS$(G, <, \tau)$

```
foreach v ∈ V(G) do label(v) ← ∅;
for i ← 1 to n do
    Eligible ← {x ∈ V(G) | x unnumbered and
    ∄y ∈ V(G) such that label(x) < label(y)};
    Let v be the leftmost vertex of Eligible according to the ordering τ;
    σ(v) ← i;
    foreach unnumbered vertex w adjacent to v do
        label(w) ← label(w) ∪ {i};
    end
end
```

Definitions of most of the searches we will consider appear in [CK08] or [Gol04] or chapter 1.

With this formalism, in order to specify a particular search we just need to specify $<$, the partial order relation on the label sets for that search. The choice of permutation τ is useful in some situations described below; otherwise, we consider the orderings output by an arbitrary choice of τ thanks to the following definition:

Definition 6.2.1. Let $<$ be some ordering over $P(\mathbb{N}^+)$. Then σ is a TBLS ordering for G and $<$ if there exists τ such that $\sigma = TBLS(G, <, \tau)$.

Before giving some examples of appropriate $<$ for well known searches, we point out some features of the TBLS formalism.

Remarks on this formalism:

1. Notice that during a TBLS search vertices are always labeled from 1 up to n. The original description of LBFS generated labels from n down to 1. Since a label is always an unordered set rather than a string, as often seen with LBFS and LDFS , we avoid having to prepend or append elements to existing labels. It should also be noticed that the TBLS model does not require the graph to be connected, and therefore in the following we will extend classical graph searches to disconnected graphs. Since we just need

to specify $<$ to describe a particular search no implementation details have to be discussed in the specification of the search. Finally, by requiring a tie-breaking permutation τ we immediately have a mechanism for choosing a specific vertex from *Eligible*. Many existing recognition algorithms such as the unit interval recognition algorithm in [Cor04b] use a series of LBFS sweeps where ties are broken by choosing the rightmost eligible vertex in the previous LBFS search; to accomplish this in the TBLS formalism τ is set to be the reverse of the previous LBFS ordering.

2. Note that all elements of the set *Eligible* must have a label set that is maximal with respect to set inclusion over some finite partially ordered set; therefore *Eligible* cannot be empty. In the context of LBFS the *Eligible* set is often called a *slice*. The reader should be aware that we make no claims on the complexity of computing the strict partial order $<$ over the label-set $P(\mathbb{N}^+)$; it could be NP-hard or even worse.

3. Our formalism allows easy determination of whether specific search A is a restriction of specific search B in the sense that all possible search orderings produced by search A could also have been produced by search B. In particular, we just have to prove that $<_A$ is an extension of $<_B$ (in the usual sense of strict orders extension); see section 6.5.

6.3 Characterizing classical searches using TBLS

In this section we show how various classical searches (see [CK08] for the definitions of these various searches) may be expressed in the TBLS formalism. In each case we will state an appropriate $<$ order and where applicable, we will establish characterizations of the search with the "pattern-condition" presented in [CK08] and recalled in chapter 1. In some cases we will exhibit a new vertex ordering characterization.

First we define some terminology. Let $N_\sigma(u, v)$ be the set of neighbors of u that occur before v in σ; formally $N_\sigma(u, v) = \{i | \sigma^{-1}(i) \in N(u) \text{ and } \sigma^{-1}(i) <_\sigma v\}$. For $A \in P(\mathbb{N}^+)$, let $umin(A)$ be: if $A = \emptyset$ then $umin(A) = \infty$ else $umin(A) =$

$min\{i|i \in A\}$; and let $umax(A)$ be: if $A = \emptyset$ then $umax(A) = 0$ else $umax(A) = max\{i|i \in A\}$.

The following lemma plays a key role for most of the proofs in this section:

Lemma 6.3.1. *Let S be a TBLS search with partial order relation $<_S$.*

An ordering σ of the vertices of a graph G is an S-ordering if and only if for every $x,\ y \in V(G)$, if $x <_\sigma y$, then $N_\sigma(x,x) \not<_S N_\sigma(y,x)$.

Proof. The forward direction follows directly from the definition.

For the reverse direction, assume that for every $x,\ y \in V(G)$, $x <_\sigma y$, $N_\sigma(x,x) \not<_S N_\sigma(y,x)$ but σ is not an S ordering. Let $\gamma = TBLS(G,S,\sigma)$. Since σ is not an S-ordering, we know that $\sigma \neq \gamma$. Now let i be the first index such that $\sigma^{-1}(i) \neq \gamma^{-1}(i)$. Let $x = \sigma^{-1}(i)$ and $y = \gamma^{-1}(i)$. Since $x <_\sigma y$ but $TBLS(G,S,\sigma)$ did not choose x, we know that x did not have a maximal label at step i. Therefore there must exist z such that $x <_\sigma z$, and $label_i(x) <_S label_i(z)$. But since $label_i(x) = N_\sigma(x,x)$ and $label_i(z) = N_\sigma(z,x)$ we now have a pair of vertices x, z that contradicts the assumption that for every $x,\ y \in V(G)$, $x <_\sigma y$, $N_\sigma(x,x) \not<_S N_\sigma(y,x)$.

□

6.3.1 Generic Search

A *generic search* as described by Tarjan [Tar72] is any search that wherever possible visits neighbors of already visited vertices (this corresponds to the usual notion of graph search).

We now give an alternate proof based on our formalism of the characterization of a generic search ordering.

Theorem 6.3.2 (see [CK08])**.** *We define $A <_{gen} B$ if and only if $A = \emptyset$ and $B \neq \emptyset$ and let σ be a permutation of $V(G)$. The following conditions are equivalent:*

1. *σ is a generic search ordering of $V(G)$ (a TBLS using $<_{gen}$).*
2. *For every triple of vertices $a,\ b,\ c$ such that $a <_\sigma b <_\sigma c$, $a \in N(c) - N(b)$ there exists $d \in N(b)$ such that $d <_\sigma b$.*

Proof. Suppose that σ is a generic search ordering. Using Lemma 6.3.1 on σ, we know that:
σ is a generic ordering
$\iff$ for every $x,\ y \in V(G)$, $x <_\sigma y$, $N_\sigma(x,x) \not<_{gen} N_\sigma(y,x)$
$\iff$ for every $x,\ y \in V(G)$, $x <_\sigma y$, $N_\sigma(y,x) = \emptyset$ or $N_\sigma(x,x) \neq \emptyset$
$\iff$ for every $x,\ y \in V(G)$, $x <_\sigma y$, $N_\sigma(y,x) \neq \emptyset \to N_\sigma(x,x) \neq \emptyset$
$\iff$ for every triple of vertices $a,\ b,\ c$ such that $a <_\sigma b <_\sigma c$, $a \in N(c) - N(b)$, there exists $d \in N(b)$ such that $d <_\sigma b$.

□

6.3.2 BFS (Breadth-First Search)

We now focus on BFS.

Theorem 6.3.3. *We define $A <_{BFS} B$ if and only if $umin(A) > umin(B)$. Let σ be a permutation of $V(G)$. The following conditions are equivalent:*

1. *σ is a BFS ordering (a TBLS using $<_{BFS}$).*
2. *for every triple $a, b, c \in V(G)$ such that $a <_\sigma b <_\sigma c$, $a \in N(c) - N(b)$, there exists d such that $d \in N(b)$ and $d <_\sigma a$.*
3. *for every triple $a, b, c \in V(G)$ such that $a <_\sigma b <_\sigma c$ and a is the leftmost vertex of $N(b) \cup N(c)$ in σ, we have $a \in N(b)$.*

Proof. The equivalence between condition (1) and condition (2) has been proved in [CK08]. Let now prove that condition (1) is equivalent to condition (3). Suppose that σ is a BFS ordering. Using Lemma 6.3.1 on σ, we know that:
σ is a BFS ordering
$\iff$ for every $x,\ y \in V(G)$, $x <_\sigma y$, $N_\sigma(x,x) \not<_{BFS} N_\sigma(y,x)$
$\iff$ for every $x,\ y \in V(G)$, $x <_\sigma y$, $umin(N_\sigma(x,x)) \not> umin(N_\sigma(y,x))$ $\iff$ for every $x,\ y \in V(G)$, $x <_\sigma y$, $umin(N_\sigma(x,x)) \leq umin(N_\sigma(y,x))$
$\iff$ for every triple of vertices $a,\ b,\ c$ such that $a <_\sigma b <_\sigma c$, a is the leftmost vertex of $N(b) \cup N(c)$, we have $a \in N(b)$.

□

6.3.3 DFS (Depth First Search)

We now turn our attention to Depth First Search.

Theorem 6.3.4. *We define $A <_{DFS} B$ if and only if $umax(A) < umax(B)$. Let σ be a permutation of $V(G)$. The following conditions are equivalent:*

1. *σ is a DFS ordering (a TBLS using $<_{DFS}$).*
2. *for every triple of vertices $a,\ b,\ c$ such that $a <_\sigma b <_\sigma c$, $a \in N(c) - N(b)$ there exists $d \in N(b)$ such that $a <_\sigma d <_\sigma b$.*
3. *for every triple of vertices $a,\ b,\ c$ such that $a <_\sigma b <_\sigma c$, and a is the rightmost vertex of $N(b) \cup N(c)$ in σ, we have $a \in N(b)$.*

Proof. The equivalence between condition (1) and (2) has been proved in [CK08]. Let us show the equivalence between (1) and (3). Suppose that σ is a DFS ordering. Using Lemma 6.3.1 on σ, we know that:
σ is a DFS ordering
$\iff$ for every $x,\ y \in V(G)$, $x <_\sigma y$, $N_\sigma(x,x) \not<_{DFS} N_\sigma(y,x)$
$\iff$ for every $x,\ y \in V(G)$, $x <_\sigma y$, $umax(N_\sigma(x,x)) \not< umax(N_\sigma(y,x))$
$\iff$ for every $x,\ y \in V(G)$, $x <_\sigma y$, $umax(N_\sigma(y,x)) \leq umax(N_\sigma(x,x))$
$\iff$ for every triple of vertices $a,\ b,\ c$ such that $a <_\sigma b <_\sigma c$ and a is the rightmost vertex of $N(b) \cup N(c)$ in σ, we have $a \in N(b)$.

□

6.3.4 LBFS (Lexicographic Breadth First Search)

Let us now consider LBFS .

Theorem 6.3.5. *We define $A <_{LBFS} B$ if and only if $umin(B-A) < umin(A-B)$. Let σ be a permutation of $V(G)$. The following conditions are equivalent:*

1. *σ is a LBFS ordering (a TBLS using $<_{LBFS}$)*
2. *for every triple $a, b, c \in V(G)$ such that $a <_\sigma b <_\sigma c$, $a \in N(c) - N(b)$, there exists $d <_\sigma a$, $d \in N(b) - N(c)$.*

3. *for every triple $a, b, c \in V(G)$ such that $a <_\sigma b <_\sigma c$ and a is the leftmost vertex of $N(b) \triangle N(c)$ in σ, then $a \in N(b) - N(c)$.*

Proof. The equivalence between (1) and (2) is well known, see [RTL76, Gol04, BDN97]. We now prove the equivalence between (1) and (3). Suppose that σ is a LBFS ordering. Using Lemma 6.3.1 on σ, we know that:
σ is a LBFS ordering
$\iff$ for every $x,\ y \in V(G)$, $x <_\sigma y$, $N_\sigma(x,x) \not<_{LBFS} N_\sigma(y,x)$
$\iff$ for every $x,\ y \in V(G)$, $x <_\sigma y$, $umin(N_\sigma(y,x) - N_\sigma(x,x)) \not< umin(N_\sigma(x,x) - N_\sigma(y,x))$
$\iff$ for every $x,\ y \in V(G)$, $x <_\sigma y$, $umin(N_\sigma(x,x) - N_\sigma(y,x)) \leq umin(N_\sigma(y,x) - N_\sigma(x,x))$
$\iff$ for every triple of vertices $a,\ b,\ c$ such that $a <_\sigma b <_\sigma c$ and a is the leftmost vertex of $N(b) \triangle N(c)$ in σ, we have $a \in N(b) - N(c)$.

□

6.3.5 LDFS (Lexicographic Depth First Search)

The Lexicographic Depth First Search (LDFS) was introduced in [CK08].

Theorem 6.3.6. *We define $A <_{LDFS} B$ if and only if $umax(A-B) < umax(B-A)$. Let σ be a permutation of $V(G)$. The following conditions are equivalent:*

1. *σ is a LDFS ordering (a TBLS using $<_{LDFS}$)*
2. *for every triple $a, b, c \in V(G)$ such that $a <_\sigma b <_\sigma c$, $a \in N(c) - N(b)$, there exists $a <_\sigma d <_\sigma b$, $d \in N(b) - N(c)$.*
3. *for every triple $a, b, c \in V(G)$ such that $a <_\sigma b <_\sigma c$ and a is the rightmost vertex in $N(b) \triangle N(c)$ in σ, $a \in N(b) - N(c)$.*

Proof. The equivalence between (1) and (2) is well known, see [CK08]. We now prove the equivalence between (1) and (3). Suppose that σ is a LDFS ordering. Using Lemma 6.3.1 on σ, we know that:
σ is a LDFS ordering
$\iff$ for every $x,\ y \in V(G)$, $x <_\sigma y$, $N_\sigma(x,x) \not<_{LDFS} N_\sigma(y,x)$
$\iff$ for every $x,\ y \in V(G)$, $x <_\sigma y$, $umax(N_\sigma(x,x) - N_\sigma(y,x)) \not< umax(N_\sigma(y,x) -$

$N_\sigma(x,x))$
$\iff$ for every $x,\ y \in V(G)$, $x <_\sigma y$, $umax(N_\sigma(y,x) - N_\sigma(x,x)) \leq umax(N_\sigma(x,x) - N_\sigma(y,x))$
$\iff$ for every triple of vertices $a,\ b,\ c$ such that $a <_\sigma b <_\sigma c$ and a is the rightmost vertex of $N(b) \triangle N(c)$ in σ, we have $a \in N(b) - N(c)$.

□

The symmetry between BFS and DFS (respectively LBFS and LDFS) becomes clear when using the TBLS ordering formalism. This symmetry was also clear using the pattern-conditions as introduced in [CK08], and in fact lead to the discovery of LDFS. To finish with classical searches, we notice that **Maximal Cardinality Search (MCS)** as introduced in [TY84], can easily be defined using the partial order: $A <_{MCS} B$ if and only if $|A| < |B|$.

Similarly **MNS (Maximal Neighborhood Search)** as introduced in [Shi84] for chordal graph recognition, is a search such that $A <_{MNS} B$ if and only if $A \subsetneq B$, i.e., it uses the strict inclusion order between subsets.

6.3.6 Limitations of the TBLS model

To finish, let us remark that there exits at least one known search that does not fit into the TBLS model. In the following, recall that $label_i(v)$ for a vertex v denotes the label of v at step i. *Layered Search* starts at a vertex s, and ensures that if $dist(s,x) < dist(s,y)$ then $\sigma(x) < \sigma(y)$. In other words it respects the layers (vertices at the same distance from the start vertex s). We now show that this search is not a TBLS by considering the graph G in figure 6.1. Assume that we have started the Layered Search with $x_1,\ x_2,\ x_3,\ x_4$ and so $label_5(x_5) = \{3\}$ and $label_5(x_6) = \{4\}$. In a layered search, both x_5 and x_6 must be eligible at step 5. Thus we must have neither $\{3\} < \{4\}$ nor $\{4\} < \{3\}$; they are incomparable labels. But now consider graph H in figure 6.1 and assume that again we have started the search with $v_1,\ v_2,\ v_3,\ v_4$. So we have $label_5(v_5) = \{3\}$ and $label_5(v_6) = \{4\}$. But in this graph we have to visit v_5 before v_6. Therefore we must have $\{3\} < \{4\}$. As a conclusion, for Layered Search the labels are not sufficient and so this search cannot be written in our formalism. The same seems

true for Min-LexBFS as defined in [Mei05] and Right Most Neighbor as used in [CDH13].

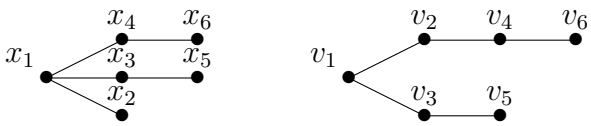


Figure 6.1: Graph G on the left and H on the right.

6.4 Two new lexicographic searches

In this section, we present two new searches based on the symmetry with LBFS and LDFS. They have been first studied in [Dus10]. In LBFS, we append numbers from n to 1 and in LDFS, we prepend numbers from 1 to n. We now study the search that appends numbers from 1 to n (LexUP) and the search that prepends numbers from n to 1 (LexDOWN).

6.4.1 LexUP

We start by presenting LexUP under the form of an algorithm.

Algorithm 13: LexUP

Data: $G = (V(G), E(G))$ an undirected graph
Result: an ordering σ of the vertices of G

```
assign label ϵ to all vertices;
for (i = 1 to |V(G)|) do
    pick any unnumbered vertex u with lexicographically largest label;
    σ(i) ← u % {give to u the number i}%;
    foreach (unnumbered vertex v ∈ N(u)) do
        append i to label(v);
    end
end
Output σ;
```

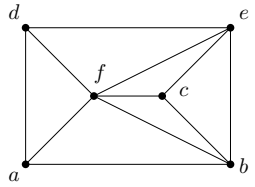


Figure 6.2: A graph G

First, consider the graph in Figure 6.2. A LexUP ordering is for example a, b, c, e, d, f.

Let us now express LexUP under the TBLS formalism by defining $<_{LexUP}$. In LexUP, a label is a word composed of numbers in increasing order. So for two labels l_a, l_b built by LexUP, l_a is lexicographically less than l_b if and only if l_a is a strict prefix of l_b or the first letter in the labels they differ on is less in l_a. So $A <_{LexUP} B$ if and only if $((A \subsetneq B) and (umax(A) < umin(B - A)))$ or $((umin(A - B) < umin(B - A))$ and $(umin(A - B) < umax(B)))$.

Theorem 6.4.1. *We define $A <_{LexUP} B$ if and only if $((A \subsetneq B)$ and $(umax(A) < umin(B - A)))$ or $((umin(A - B) < umin(B - A))$ and $(umin(A - B) < umax(B)))$. Let σ be a permutation of $V(G)$. The following conditions are equivalent:*

1. *σ is a LexUP ordering (a TBLS using $<_{LexUP}$)*
2. *for every $a,\ b,\ c \in V(G)$ such that $a <_\sigma b <_\sigma c$ and a is the leftmost vertex of $N(b) \triangle N(c)$, if $ac \in E$ then there exists $d \in V(G)$, $a <_\sigma d <_\sigma b$, $db \in E(G)$ and if $ab \in E(G)$ then there is no vertex d such that $a <_\sigma d <_\sigma b$, $dc \in E(G)$.*

Proof. Suppose that σ is a LexUP ordering. Using Lemma 6.3.1 on σ, we know that:
σ is a LexUP ordering
$\iff$ for every $x,\ y \in V(G)$, $x <_\sigma y$, $N_\sigma(x, x) \not<_{LexUP} N_\sigma(y, x)$
$\iff$ for every $x,\ y \in V(G)$, $x <_\sigma y$, $N_\sigma(y, x) = N_\sigma(x, x)$ or $N_\sigma(y, x) <_{LexUP} N_\sigma(x, x)$

$\iff$ for every $x,\ y \in V(G),\ x <_\sigma y,\ N_\sigma(y,x) = N_\sigma(x,x)$ or $((N_\sigma(y,x) \subsetneq N_\sigma(x,x))$ and $(umax(N_\sigma(y,x)) < umin(N_\sigma(x,x) - N_\sigma(y,x))))$ or $((umin(N_\sigma(y,x) - N_\sigma(x,x)) < umin(N_\sigma(x,x) - N_\sigma(y,x)))$ and $(umin(N_\sigma(y,x) - N_\sigma(x,x)) < umax(N_\sigma(x,x))))$.

It should be noted that $umin(N_\sigma(y,x) - N_\sigma(x,x)) < umin(N_\sigma(x,x) - N_\sigma(y,x))$ is equivalent to the leftmost vertex of $N_\sigma(y,x) \triangle N_\sigma(x,x)$ belongs to $N_\sigma(y,x)$ and $N_\sigma(y,x) \subsetneq N_\sigma(x,x)$ implies that the leftmost vertex of $N_\sigma(y,x) \triangle N_\sigma(x,x)$ belongs to $N_\sigma(x,x)$.

$\iff$ for every $a,\ b,\ c \in V(G)$ such that $a <_\sigma b <_\sigma c$ and a is the leftmost vertex of $N(b) \triangle N(c)$, if $ac \in E$ then there exists $d \in V(G)$, $a <_\sigma d <_\sigma b$, $db \in E(G)$ and if $ab \in E(G)$ then there is no vertex d such that $a <_\sigma d <_\sigma b$, $dc \in E(G)$. □

6.4.2 LexDOWN

We now consider LexDOWN and start by presenting LexDOWN under the form of an algorithm.

Algorithm 14: LexDOWN

Data: $G = (V(G), E(G))$ an undirected graph
Result: an ordering σ of the vertices of G

```
assign label ϵ to all vertices;
for (i = 1 to |V(G)|) do
    pick any unnumbered vertex u with lexicographically largest label;
    σ(i) ← u % {give to u the number i}%;
    foreach (unnumbered vertex v ∈ N(u)) do
        prepend |V(G)| − i to label(v);
    end
end
Output σ;
```

First, consider the graph in Figure 6.2. A LexDOWN ordering is for example a, b, d, c, f, e.

Let us now express LexDOWN under the TBLS formalism by defining $<_{LexDOWN}$. In LexDOWN, a label is a word composed of numbers in increasing order. So for

two labels l_a, l_b built by LexDOWN, l_a is lexicographically less than l_b if and only if l_a is a strict prefix of l_b or the first letter in the labels they differ on is less in l_a. So $A <_{LexDOWN} B$ if and only if $((A \subsetneq B)$ and $(umax(B - A) < umin(A)))$ or $((umax(B - A) < umax(A - B))$ and $(umin(B) < umax(A - B)))$.

Theorem 6.4.2. *We define $A <_{LexDOWN} B$ if and only if $((A \subsetneq B)$ and $(umax(B-A) < umin(A)))$ or $((umax(B-A) < umax(A-B))$ and $(umin(B) < umax(A - B)))$. Let σ be a permutation of $V(G)$. The following conditions are equivalent:*

1. *σ is a LexDOWN ordering (a TBLS using $<_{LexDOWN}$)*

2. *for every $a,\ b,\ c \in V(G)$ such that $a <_\sigma b <_\sigma c$ and a is the rightmost vertex of $N(b) \triangle N(c)$, if $ac \in E$ then there exists $d \in V(G)$, $d <_\sigma a$, $db \in E(G)$ and if $ab \in E(G)$ then there is no vertex d such that $d <_\sigma a$, $dc \in E(G)$.*

Proof. Suppose that σ is a LexDOWN ordering. Using Lemma 6.3.1 on σ, we know that:
σ is a LexDOWN ordering
$\iff$ for every $x,\ y \in V(G)$, $x <_\sigma y$, $N_\sigma(x,x) \not<_{LexDOWN} N_\sigma(y,x)$
$\iff$ for every $x,\ y \in V(G)$, $x <_\sigma y$, $N_\sigma(x,x) = N_\sigma(y,x)$ or $N_\sigma(y,x) <_{LexDOWN} N_\sigma(x,x)$
$\iff$ for every $x,\ y \in V(G)$, $x <_\sigma y$, $N_\sigma(x,x) = N_\sigma(y,x)$ or $((N_\sigma(y,x) \subsetneq N_\sigma(x,x))$ and $(umax(N_\sigma(x,x) - N_\sigma(y,x)) < umin(N_\sigma(y,x))))$ or $((umax(N_\sigma(x,x) - N_\sigma(y,x)) < umax(N_\sigma(y,x) - N_\sigma(x,x)))$ and $(umin(N_\sigma(x,x)) < umax(N_\sigma(y,x) - N_\sigma(x,x))))$

It should be noted that $umax(N_\sigma(x,x) - N_\sigma(y,x)) < umax(N_\sigma(y,x) - N_\sigma(x,x))$ is equivalent to the rightmost vertex of $N_\sigma(y,x) \triangle N_\sigma(x,x)$ belongs to $N_\sigma(y,x)$ and $N_\sigma(y,x) \subsetneq N_\sigma(x,x)$ implies that the rightmost vertex of $N_\sigma(y,x) \triangle N_\sigma(x,x)$ belongs to $N_\sigma(x,x)$.

$\iff$ for every $a,\ b,\ c \in V(G)$ such that $a <_\sigma b <_\sigma c$ and a is the rightmost vertex of $N(b) \triangle N(c)$, if $ac \in E$ then there exists $d \in V(G)$, $d <_\sigma a$, $db \in E(G)$ and if $ab \in E(G)$ then there is no vertex d such that $d <_\sigma a$, $dc \in E(G)$. $\square$

6.5 TBLS hierarchy of searches

Here we explain how hierarchies of searches may be built using the TBLS formalism. There are two natural ways of saying that a search S **is a** search S' (for instance, that LBFS **is a** BFS): either the $<$ ordering used by S is a refinement of that of S'; or any search ordering σ output by an execution of S could also have been output by an execution of S'. We start with the definition of an extension of a TBLS search.

Definition 6.5.1. For two TBLS searches S, S', we say that S' is an extension of S (denoted by $S \ll S'$) if and only if every S'-ordering σ also is an S-ordering.[1]

The statement and proof of the next lemma follows the work of [KSB11] where there are similar results for the GLS formalism.

Lemma 6.5.2 (see [KSB11])**.** *For any TBLS S, any integer $p \geq 1$ and any sets A and B of $P(\mathbb{N}_p^+)$, if $A \not<_S B$ then there exists a graph G and an S-ordering σ such that in the $p-1$st step the label of the $p-1$st vertex $= A$ and the label of the pth vertex $= B$ (i.e., $label_{p-1}(\sigma^{-1}(p-1)) = A$ and $label_{p-1}(\sigma^{-1}(p)) = B$).*

Proof. Let $G = (V(G), E(G))$, with $V(G) = \{z_1, ..., z_p\}$ and $E(G) = \{z_i z_k | 1 \leq k < i \leq p-2$ and if $A \cap P(\mathbb{N}_i^+) <_S B \cap P(\mathbb{N}_i^+)$ then $k \in B$ else $k \in A\} \cup \{z_{p-1} z_k | k \in A\} \cup \{z_p z_k | k \in B\}$. Let $\sigma = (z_1, \ldots, z_p)$. By the definitions of $E(G)$ and σ, for any integers i, j such that $1 \leq i \leq j \leq p$, $N_\sigma(\sigma^{-1}(j), \sigma^{-1}(i)) = A \cap P(\mathbb{N}_i^+)$ or $N_\sigma(\sigma^{-1}(j), \sigma^{-1}(i) = B \cap P(\mathbb{N}_i^+)$ with $N_\sigma(\sigma^{-1}(i), \sigma^{-1}(i)) \not<_S N_\sigma(\sigma^{-1}(j), \sigma^{-1}(i))$. By Lemma 6.3.1, σ is an S ordering and we have $N_\sigma(\sigma^{-1}(p-1), \sigma^{-1}(p-1)) = A$ and $N_\sigma(\sigma^{-1}(p), \sigma^{-1}(p-1)) = B$. □

The relationships amongst graph searches has been a main issue in the articles [CK08, KSB11]. But in both articles, proving relationships amongst graph searches required properties on those graph searches. The next theorem shows how to exhibit such relationships without requiring any particular property beyond the $<$ definitions of the searches.

[1] This definition is consistent with the usual extension ordering used in partial order theory; in particular $S \ll S'$ means that the set of comparabilities in S is included in those of S'.

Definition 6.5.3. For two partial orders $<_P, <_Q$ on the same ground set X, we say that P extends Q if $\forall x, y \in X$, $x <_Q y$ implies $x <_P y$.

Theorem 6.5.4. *Let S, S' be two TBLS. S' is an extension of S if and only if $<_{S'}$ is an extension of $<_S$.*

Proof. For the forward direction, assume for contradiction that S' is an extension of S but $<_{S'}$ is not an extension of $<_S$. Therefore there exists A, B such that $A <_S B$ and $A \not<_{S'} B$. Now using the previous lemma there exists a graph G and an S'-ordering σ such that $label_{p-1}(\sigma^{-1}(p-1)) = A$ and $label_{p-1}(\sigma^{-1}(p)) = B$. Since $A <_S B$ using Lemma 6.3.1, we deduce that σ is not an S-ordering which contradicts the fact that S' is an extension of S.

Suppose that σ is an S'-ordering. Therefore using Lemma 6.3.1 we know that for every $x,\ y \in V(G)$, $x <_\sigma y$, $N_\sigma(x,x) \not<_{S'} N_\sigma(y,x)$. Since $<_{S'}$ is an extension of $<_S$, we deduce that $x,\ y \in V(G)$, $x <_\sigma y$, $N_\sigma(x,x) \not<_S N_\sigma(y,x)$. Now using the previous theorem, we deduce that σ is an S-ordering.

□

We now use this theorem to easily rediscover the relationships amongst various graph searches as noted in [CK08] and [KSB11].

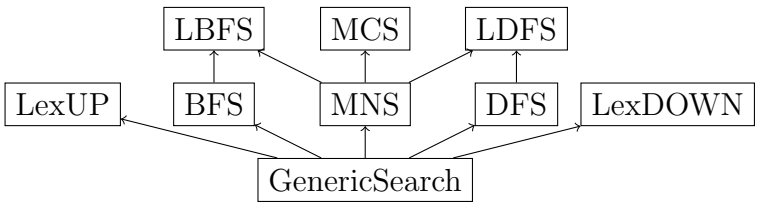


Figure 6.3: Summary of the heredity relationships proved in theorem 6.5.5. An arc from Search S to search S' means that S' extends S.

Theorem 6.5.5. *The partial order of the relation extension between classical searches is described in Figure 1.*

Proof. To show that a search extends another one we will use theorem 6.5.4.

Let us show that $<_{BFS}$ (respectively $<_{DFS}$, $<_{MNS}$) is an extension of $<_{gen}$. Let $A <_{gen} B$. By definition we have $A = \emptyset$ and $B \neq \emptyset$. As a consequence we have

$umin(B) < umin(A)$ and thus $A <_{BFS} B$ (respectively $umax(A) < umax(B)$ implying $A <_{DFS} B$, and $A \subsetneq B$ implying $A <_{MNS} B$).

Let us show that $<_{LexUP}$ (respectively $<_{LexDOWN}$) is an extension of $<_{gen}$. Let $A <_{gen} B$. By definition we have $A = \emptyset$ and $B \neq \emptyset$. As a consequence we have $(umax(A) < min(B - A))$ and thus $A <_{LexUP} B$ (respectively $(umax(B - A) < umin(A))$ implying $A <_{LexDOWN} B$).

We now show that $<_{LBFS}$ is an extension of $<_{BFS}$. Let $A <_{BFS} B$. By definition, we have $umin(B) < umin(A)$. As a consequence, $umin(B - A) < umin(A - B)$, implying $A <_{LBFS} B$.

To see that $<_{LDFS}$ is an extension of $<_{DFS}$, first suppose that $A <_{DFS} B$. Therefore $umax(A) < umax(B)$ and as a consequence $umax(A-B) < umax(B-A)$ thereby implying $A <_{LDFS} B$.

Similarly $<_{LBFS}$ (respectively $<_{LDFS}$, $<_{MCS}$) is an extension of $<_{MNS}$. Let $A <_{MNS} B$; by definition we have $A \subsetneq B$. As a consequence, $umin(B - A) < umin(A - B)$ (respectively $umax(A - B) < umax(B - A)$ and $|A| \leq |B|$). So $A <_{LBFS} B$ (respectively $A <_{LBFS} B$ and $A <_{MCS} B$).

□

6.6 The relationship between GLS and TBLS

We are now interested in determining the relationship between TBLS and GLS. First let us recall GLS from [KSB11]. It depends on a *labeling structure* which consists of four elements:

- a set of labels L;
- a strict order $<_{GLS}$ over the label-set L;
- an initial label l_0;
- an UPLAB function $L \times \mathbb{N} \to L$.

The GLS algorithm then takes as input a graph $G = (V(G), E(G))$ (over which the search is performed) as well as a labeling structure.

The computational power of the UPLAB function is unbounded, even though it must be deterministic, and the label set L may be any set. In contrast, TBLS

uses no initial label, a fixed label set $P(\mathbb{N}^+)$, and a fixed simple updating function. Despite these restrictions, it is, however equivalent to GLS in the sense of theorem 6.6.3.

Algorithm 15: GLS($G, \{L, <_{GLS}, l_0, UPLAB\}$)

```
foreach v ∈ V(G) do l(v) ← l_0;
for i ← 1 to n do
    Let Eligible be the set of eligible vertices, i.e., those with l(v) maximal
    with respect to <_GLS;
    Let v be some vertex from Eligible;
    σ(i) ← v;
    foreach unnumbered vertex w adjacent to v do
        l(w) ← UPLAB(l(w), i);
    end
end
```

We now prove that for each GLS, there is a $<_{GTBS}$ producing the same orderings, and conversely. First we establish some notation.

At each iteration i of $TBLS(G, <_{TBLS}, \sigma)$, let $l_{TBLS,i}(v)$ be the label assigned to v by $TBLS(G, <_{TBLS}, \sigma)$, i.e., the label that will be used to choose the ith vertex. Similarly, let $l_{GLS,i}(v)$ be the label assigned to v by GLS($G, \{L, <_{GLS}, l_0, UPLAB\}$), i.e., the label that will be used to choose the ith vertex. Given a graph $G = (V(G), E(G))$, and an ordering σ of $V(G)$, let us define $I_k^\sigma(v) = N(v) \cap \{\sigma(1), \ldots \sigma(k)\}$ to be the neighbours of v visited at step k. Let us define p_k^j to be the j-th element of $I_k^\sigma(v)$ sorted in increasing visiting order.

Proposition 6.6.1 ([KSB11])**.** *Let S be a labeling structure, $G = (V(G), E(G))$ a graph, and $v \in V(G)$. At iteration i of GLS(G, S),*
$l_{GLS,i}(v) = UPLAB(UPLAB(\ldots UPLAB(l_o, p_{i-1}^1) \ldots, p_{i-1}^{|I_k^\sigma(v)|-1}), p_{i-1}^{|I_k^\sigma(v)|})$

Proposition 6.6.2. *Let $G = (V(G), E(G))$ be a graph, $v \in V(G)$, and σ the order produced by TBLS($G, <, \tau$). At iteration i of TBLS($G, <, \tau$), $l_{TBLS,i}(v) = I_{i-1}^\sigma(v)$.*

Proof. The proof goes by induction. At the first step of the algorithm, every vertex has $\varnothing$ as its label, and has no previously visited neighbor.

Assume that at iteration i, every unnumbered vertex x has label $l_{TBLS,i}(x) = I^{\sigma}_{i-1}(x)$. After this iteration, for every neighbor v of $\sigma(i)$, $l_{TBLS,i}(v) = l_{TBLS,i-1}(v) \cap \{i\}$, which is indeed $I^{\sigma}_{i}(v)$, and for every non-neighbor v of $\sigma(i)$, $l_{TBLS,i}(v) = l_{TBLS,i-1}(v)$, which is again $I^{\sigma}_{i}(v)$.

□

Theorem 6.6.3. *A set T of orderings of the vertices of a graph G is equal to $\{TBLS(G, <_{TBLS}, \tau) \text{ — } \tau \in \mathfrak{S}_n\}$ if and only if there exists a labeling structure $S = (L, <_{GLS}, l_0, UPLAB)$ such that T is equal to the set of orderings produced by $GLS(G, S)$.*

Proof. First, consider an order $<_{TBLS}$. $\{TBLS(G, <_{TBLS}, \tau) \text{ — } \tau \in \mathfrak{S}_n\}$ is equal to the set of all orderings produced by GLS(G, S) with $S = (P(\mathbb{N}^+), <_{TBLS}, \varnothing, cons)$, where $cons(l(w), i)$ returns $l(w) \cup \{i\}$.

Conversely, consider $S = (L, <_{GLS}, l_0, UPLAB)$ a labeling structure. We show that there exists an order $<_{TBLS}$ such that, for every graph G, the set of all orderings produced by GLS(G, S) is equal to $\{TBLS(G, <_{TBLS}, \tau) \text{ — } \tau \in \mathfrak{S}_n\}$.

By propositions 6.6.1 and 6.6.2 we can define an injection ϕ from $P(\mathbb{N}^+)$ (the labels used by TBLS) into labels *effectively* used by GLS (i.e. those who can be worn by a vertex during some execution of the algorithm). ϕ is recursively defined as $\phi(\varnothing) = l_0$, and if $max(A) = i$, then $\phi(A) = \text{UPLAB}(\phi(A) \backslash \{i\}, i)$. Notice the same GLS-label l may be reached in different ways. $\phi^{-1}(l) \subseteq P(\mathbb{N}^+)$ is the set of TBLS-labels that correspond to that label. It is empty for all labels not effectively used.

Then, we define $<_{TBLS}$ as follows: $\forall l, l' \in L$ such that $l = \phi(A)$ and $l' = \phi(A')$, $l <_{GLS} l'$ if and only if $A <_{TBLS} A'$. We are now ready to prove the theorem. The proof goes by induction. Before the first iteration, for every vertex v, $l^0_{GLS}(v) = l_0$ and $l^0_{TBLS}(v) = \varnothing$. GLS can pick any of these vertices, in particular the one that would be picked by $TBLS(G, <_{TBLS}, \tau)$, and by setting τ to be equal to a given output of $GLS(G, S)$, TBLS would indeed chose the same vertex.

Now, assume that when step i begins, both algorithms have produced the order $\sigma(1) \ldots \sigma(i-1)$, and the ith vertex is about to be chosen. By propositions 6.6.1 and 6.6.2, for every unnumbered vertex x, $l_{TBLS,i}(x) = I^{\sigma}_{i-1}(x)$, and $l_{GLS,i}(x) = l_{GLS,i}(v) = UPLAB(\ldots UPLAB(l_o, p^1_{i-1}) \ldots, p^{|I^{\sigma}_k(v)|}_{i-1})$. By the definition of ϕ, we have that $l_{GLS,i}(x) = \phi(l_{TBLS,i}(x))$, and $l_{TBLS,i}(x) \in \phi^{-1}(l_{GLS,i}(x))$.

Then, by the definition of $<_{TBLS}$, for two unnumbered vertex v and w, we know that $l_{TBLS,i}(v) <_{TBLS} l_{TBLS,i}(w)$ if and only if $\phi(l_{TBLS,i}(v)) <_{GLS} \phi(l_{TBLS,i}(w))$, and $l_{GLS,i}(v) <_{GLS} l_{GLS,i}(w)$ if and only if for all $l_v \in \phi^{-1}(l_{GLS,i}(v))$ and all $l_w \in \phi^{-1}(l_{GLS,i}(w))$, $l_v <_{TBLS} l_w$. Thus, the set of eligible vertices at step i is the same for both algorithms. GLS can pick any of these vertices, in particular the one that would be picked by $TBLS(G, <_{TBLS}, \tau)$, and by setting τ to be equal to $GLS(G, S)$, TBLS would indeed choose the right vertex. □

Although TBLS and GLS cover the same set of vertex orderings, we think that our TBLS formalism provides a simpler framework to analyse multisweep algorithms, as can be seen in the next section.

6.7 Test, certification and implementation

We now turn our attention to the question of testing and certifying a given search. The testing problem is: given an ordering σ and a search $<_S$, is σ an *S-ordering* (i.e., does there exist τ such that $\sigma = TBLS(G, <, \tau)$?) When this question is asked, no justification is needed and so we can use the algorithm itself. The certification problem is: given an ordering σ and a search $<_S$, justify that σ is or is not an *S-ordering*. To answer this question we cannot use the original algorithm.

The difference between the two problems is important from a practical perspective. The certification problem is concerned with the problem of trusting an implementation of a search S, contrary to the testing problem which assumes that there exists a good implementation of a search S that can be used, and simply answers yes or no, i.e., a decision problem.

6.7.1 Testing a TBLS

The main result of this section is to present an algorithm that will *test* a TBLS. Recall from Definition 6.2.1 that σ is a TBLS ordering for G and $<$ if there exists τ such that $\sigma = TBLS(G, <, \tau)$.

Theorem 6.7.1. *Let: G be a graph; $<$ a search rule; and σ an ordering of the vertices of G. Then there exists τ such that $\sigma = TBLS(G, <, \tau)$ if and only if $\sigma = TBLS(G, <, \sigma)$.*

Proof. One direction is obvious. For the other direction, assume that $\sigma = TBLS(G, <, \tau)$ for some τ, and consider $\sigma' = TBLS(G, <, \sigma)$. Assume, by contradiction, that $\sigma \neq \sigma'$, and consider i, the index of the first difference between σ and σ'. Let $Eligible_i^\sigma$ be the set of eligible vertices at step i of the algorithm that generated σ, and let $Eligible_i^{\sigma'}$ be the set of eligible vertices at step i of the algorithm that generated σ'. Since σ and σ' are equal until index i, $Eligible_i^\sigma = Eligible_i^{\sigma'}$. By the definition of $TBLS$, $\sigma(i)$ is the first vertex of $Eligible_i^\sigma$ according to τ. Finally, since the first vertex of this set, according to σ, is $\sigma(i)$, the tie-break rule chose it and so $\sigma(i) = \sigma'(i)$, a contradiction. □

The main issue with this algorithm is that in order to verify whether an ordering is indeed an S-ordering or not, one has to trust the implementation of S. A common method to avoid this problem is the use of certificates, which are witnesses that are simple to test. For a survey on certifying algorithms, we refer the reader to [MMNS11].

6.7.2 Certifying that an ordering is produced by a given search

Using Theorem 6.7.1 we can build an algorithm that tests whether or not $\sigma = TBLS(G, <, \sigma)$ for some fixed $<$. Let $\tau = TBLS(G, <, \sigma)$. If $\tau = \sigma$ then the answer is yes; otherwise it is no. We can certify the no answer using the first difference between τ and σ. Let i be the first index such that $\sigma(i) \neq \tau(i)$. If TBLS chooses $\sigma(i)$ and not $\tau(i)$ at step i, then at this time $l(\sigma(i)) < l(\tau(i))$. So we must be able to build a contradiction to the pattern-condition of this search.

The next theorem improves results proposed in [KSB11].

Theorem 6.7.2. *BFS, DFS, LBFS and LDFS have a certificate with $O(n^2)$ space complexity that can be built and tested in $O(n^2)$ time for BFS and DFS and $O(nm)$ time for LBFS and LDFS .*

Proof. To build the certificate we will use the third condition of the four relevant theorems in Section 6.3, in particular Theorems 6.3.3 (BFS), 6.3.4 (DFS), 6.3.5 (LBFS) and 6.3.6 (LDFS). All of these conditions are pattern-conditions. The certificate is stored in a table whose entries are keyed by the pair (b, c) where $b <_\sigma c$ and the information will either be the vertex a, where $a <_\sigma b$ that satisfies the corresponding condition or an error message indicating that the condition has been violated. For the non-lexicographic searches the pattern-condition examines a, the leftmost (BFS) or the rightmost (DFS) vertex of $N(b) \cup N(c)$ and requires that $a \in N(b)$. Using straightforward techniques (including sorting all adjacency lists according to σ) it can be shown that this can be accomplished in constant time, for any b and c. For the lexicographic searches, the pattern-condition examines a, the leftmost (LBFS) or the rightmost (LDFS) vertex of $N(b) \triangle N(c)$ and requires that $a \in N(b) - N(c)$. Again, it is easy to show that this can be accomplished in time $O(|N(b)| + |N(c)|$, for any b and c. In all cases, if a satisfies the membership condition then it is stored in the b, c'th entry of the table; otherwise "error" is stored.

Regarding complexity considerations, the table gives us the $\mathrm{O}(n^2)$ space complexity. For the non-lexicographic searches $O(n^2)$ time is required to build and search the table. For the lexicographic searches, the timing requirement is bounded by $\sum_{b \in V(G)} \sum_{c \in V(G)} (|N(b)| + |N(c)|)$ to build the table and $O(n^2)$ time to search it, giving an $O(nm)$ timing complexity. □

6.7.3 Implementation

It is not hard to verify that classical searches such as BFS, DFS, LBFS, LDFS, can be implemented as a TBLS search, i.e., solving the tie-break with a given total ordering τ of the vertices, within the same complexity as their current best implementations. We now prove an upper bound valid for all TBLS searches.

Theorem 6.7.3. *$TBLS(G, <, \tau)$ can be implemented to run in $O(mT(n) \log n)$ time where the $<$ comparison time between two labels by is bounded by $O(T(n))$.*

Proof. We use the framework of partition refinement [HMPV00]. First we sort the adjacency lists with respect to τ, and consider the following algorithm. The input to the algorithm is a graph $G = (V(G), E(G))$, an order $<$ on $P(\mathbb{N}^+)$, and an ordering τ of $V(G)$ and the output is the $TBLS(G, <, \tau)$-ordering σ of $V(G)$.

Algorithm 16: Computing a TBLS ordering

```
Let 𝒫 be the partition {V(G)}, where the only part V(G) is ordered with
respect to τ;
for i ← 1 to n do
    Let Eligible be the part of 𝒫 with the largest label with respect to <
    and x its first vertex;
    replace Eligible by Eligible − {x} in 𝒫 ;
    σ(i) ← x;
    Refine(𝒫, N(x));
end
```

The algorithm maintains an (unordered) partition $\mathcal{P}$ of the unvisited vertices. Each part of $\mathcal{P}$ is an ordered list of vertices. The following two invariants hold throughout the execution of the algorithm:

1. The vertices of each part have the same unique (with respect to parts) label;
2. Inside a part, the vertices are sorted with respect to τ.

The action of Refine($\mathcal{P}, A$) is to replace each part $P \in \mathcal{P}$ with two new parts: $P \cap A$ and $P - A$ (ignoring empty new parts). It is possible to maintain the two invariants using the data structure from [HMPV00], provided the adjacency lists of G are sorted with respect to τ. After each refinement, each part of $\mathcal{P}$ therefore contains vertices that are twins with respect to the visited vertices (Invariant 1). Thanks to the second invariant, the chosen vertex is always the first vertex (with respect to τ) of part *Eligible*; i.e., $\sigma(i)$ is indeed x.

For the time complexity, Refine($\mathcal{P}$, $N(x)$) takes $O(|N(x)|)$ time [HMPV00], so all refinements take $O(n + m)$ time. The only non-linear step is identifying part *Eligible* among all parts of the partition. Each part has a label (the one shared with all its vertices) used as a key in a Max-Heap. Refine($\mathcal{P}$, $N(x)$) creates at most $|N(x)|$ new parts so there are at most m insertions in the heap. The label of a part does not change over time (but empty parts must be removed). There are no fewer removal operations than insertion operations, each consisting of at most $\log n$ label comparisons (since there are at most n parts at any time). So we get the $O(mT(n) \log n)$ time bound.

□

6.8 Application: the TBLS dimension, a new graph parameter

Even though there exists searches that are not TBLSs, our model is powerful enough to express many graph algorithms. In the following, we will focus on the TBLS that are polynomially computable (i.e. the ones in which two labels can be compared in polynomial time), and let us now focus our attention on multisweep TBLS algorithms. A multisweep TBLS algorithm is an algorithm that computes a series of ordering and outputs the last ordering computed. The first ordering σ^0 is any ordering of the vertices and then the ordering σ_i is obtain by performing $TBLS(G, <_S, \sigma_{i-1})$ or $TBLS(G, <_S, (\sigma_{i-1})^-)$. For a given property $\mathcal{P}$ and a class of graph $\mathcal{C}$, let us define the TBLS dimension of $\mathcal{P}, \mathcal{C}$ (denoted by $tblsdim(\mathcal{P}, \mathcal{C})$ as the minimal number of TBLS that a multisweep TBLS algorithm needed to perform in order to compute for every graph $G \in \mathcal{C}$ an ordering that satisfy the property $\mathcal{P}$.

This new parameter measures the efficiency of graph search to compute some graph property, but it also reveals structural aspects of graph classes as it will be discussed next.

Many structured classes of graphs, such as cographs, interval, chordal ... can be characterized by the existence of an ordering of the vertices satisfying a given property, resp. interval ordering, simplicial ordering ...

It is well known that G is a chordal graph if and only if there exists an ordering σ such that for every $x <_\sigma y <_\sigma z$, if $xz,\ yz \in E(G)$ then $xy \in E(G)$. This kind of ordering will be called a simplicial elimination ordering. It has been proved in [TY84], that performing a LBFS yields a simplicial elimination ordering if and only if the graph is chordal. Therefore the TBLS dimension of σ is a simplicial elimination ordering for the class of chordal graphs is 1.

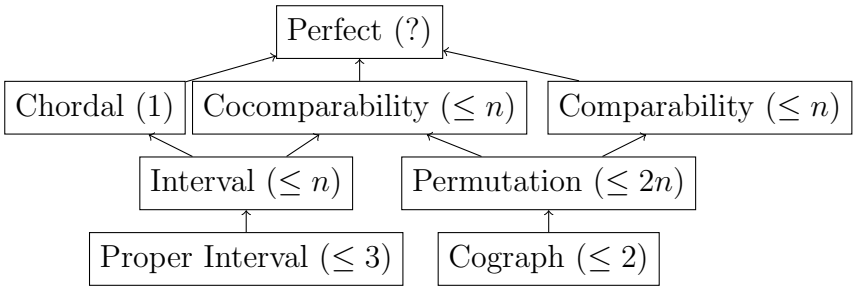
The cograph recognition algorithm presented [BCHP08] uses only one LBFS $^+$ and then one LBFS $^+$ on the complement. In the article, the authors explain how to perform a LBFS on the complement without computing the complement; this is done by taking the inverse of $<_{LBFS}$. Therefore $tblsdim$(Cograph)≤ 2.

It is well-known from [Rob71] that a graph G is a unit interval graph if and only if there exists an ordering σ such that for every $x <_\sigma y <_\sigma z$, if $xz \in E(G)$

then $xy,\ yz \in E(G)$. This kind of ordering will be called an unit interval ordering. In [Cor04b], a TBLS multisweep algorithm has been found to find an unit ordering. This algorithm uses 3 consecutive LBFS $^+$. Therefore $tblsdim$(Proper Interval)≤ 3.

In [KS93], it has been noticed that G is a cocomparability graph if and only if there exists an ordering σ such that for every $x <_\sigma y <_\sigma z$, if $xz \in E(G)$ then $xy \in E(G)$ or $yz \in E(G)$. Such an ordering is called a cocomp ordering. In chapter 5, it has been proved that performing n LBFS $^+$ always yields a cocomp ordering if the graph is a cocomparability graph. Therefore $tblsdim$(Cocomp)$\leq n$. As a consequence $tblsdim$(Permutation)$\leq 2n$ and $tblsdim$(Interval)$\leq n$. It should be noticed that we do not know any graph search based recognition algorithm for permutation graphs, therefore this bound could be far from being optimal.

The following picture sums up some known results about TBLS based recognition algorithms for well-known graph classes:



6.9 Concluding remarks

We have focused our study on a new formalism that captures many usual graph searches as well as the commonly used multi-sweep application of these searches. The TBLS formalism allows us to define a generic TBLS-orderings recognition algorithm, and gives us a new point of view of the hierarchy amongst graph searches. The new pattern-conditions for LBFS and LDFS give us a better way of certifying whether a given ordering is a LBFS or LDFS than the pattern-conditions

presented in [CK08]. Furthermore, we do not have to trust the implementation of the search (which can be complicated) but have presented a simple program that just visits the neighborhood of the vertices of the graph and stores a small amount of information (see Theorem 6.7.2). The size of this extra information, however, can be bigger than the size of the input, and it may take longer to compute than the actual time needed to perform the search itself.

While being as general as GLS, we feel that TBLS is closer to the pattern-conditions presented in [CK08], since many of the $<$ conditions presented in this chapter are a rewriting of their pattern-conditions. Still, there are many variants of the searches we studied that do not fall under the TBLS, such as layered search. We wonder if a more general search model can be found, that would not only include some of these other common searches but would also retain the simplicity of TBLS.

The landscape of graph search is quite complex. Graph searches can be clustered using the data structures involved in their best implementations (queue, stack, partition refinement ...). In this chapter we have tried a more formal way to classify graph searches. This attempt yields an algebraic framework that could be of some interest.

Clearly being an extension (see section 6.5) is a transitive relation. In fact $\ll$ structures the TBLS graph searches as $\wedge$-semilattice. The 0 search in this semilattice, denoted by the null search or S_{null}, corresponds to the empty ordering relation (no comparable pairs). At every step of S_{null} the Eligible set contains all unvisited vertices. Therefore for every τ, $TBLS(G, <_{S_{null}}, \tau) = \tau$ and so any total ordering of the vertices can be produced by S_{null}.

The infimum between two searches S, S' can be defined as follows:

For every pair of label sets A, B, we define: $A <_{S \wedge S'} B$ if and only if $A <_S B$ and $A <_{S'} B$.

Clearly every extension of S and S' is an extension of $S \wedge S'$. Similarly S and S' are extensions of $S \wedge S'$.

A

Notation

A.1 Sets and binary relations

A set S is a collection of element $x \in S$. The cardinality of a set S is denoted by $|S|$. The symbols $\subseteq$, $\subsetneq$, $\cap$, $\cup$, $-$, $\triangle$ are respectively denoted the subset, strict subset, intersection, union, difference and symmetric difference operators. The cartesian product of a set S is noted $S \times S$.

We denote by $\mathbb{N}$ the set of integers greater or equal than 0. $\mathbb{N}^+$ denotes the set of integers strictly greater than 0 and by $\mathbb{N}_p^+$ the set of integers strictly greater than 0 and less than p. $P(\{1, ..., n\})$ denotes the power-set of $\{1, ..., n\}$ and $\mathfrak{S}_n$ the set of all permutations of $\{1, ..., n\}$.

Definition A.1.1. A binary relation R on a ground set X is an ordered pair (X, E) where $E \subseteq X \times X$ and $(a, b) \in E$ iff aRb.

A.2 Graphs

A graph $G = (V(G), E(G))$ is a binary relation. In graph theory, we make the difference between directed and undirected. A graph is undirected iff $(a, b) \in E(G)$ impies $(b, a) \in E(G)$. A graph is directed if is not undirected. A loop in a graph is an edge (u, u) for some vertex u. A graph is simple if it has no loop and no multiple edges.

In this thesis, all the graphs are without loops and most of them are undirected.

For convenience, instead of denoting an edge by a set $(x_1,\ x_2)$, we denote it by the words x_1x_2 and x_2x_1. Therefore for a graph G, $E(G)$ is also considered as a set of two letter words over the alphabet $V(G)$. The complement of a simple graph $G = (V(G), E(G))$ is defined by $\overline{G} = (V(G), \overline{E(G)})$ where $\overline{E(G)} = \{(u, v) \in V(G) \times V(G) \mid u \neq v \text{ and } (u, v) \notin E(G)\}$. For a vertex v in a graph $G = (V(G), E(G))$, the neighbourhood of v is define by $N(v) = \{x \in V(G) | xv \in E(G)\}$. The variable n is used to denote the number of vertices of G, i.e. the cardinality of V(G), and m is used to denote the number of edges of G, i.e. the cardinality of E(G). An example of a graph is given in figure A.1.

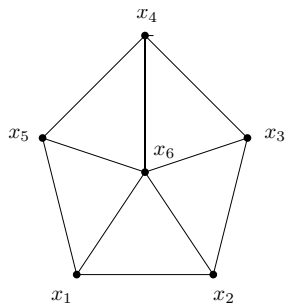


Figure A.1: A simple undirected graph G where $V(G) = \{x_1, x_1, x_2, x_3, x_4, x_5, x_6\}$ and $E(G) = \{x_1x_2,\ x_2x_3,\ x_3x_4,\ x_4x_5,\ x_1x_5,\ x_1x_6,\ x_2x_6,\ x_3x_6,\ x_4x_6,\ x_5x_6\}$

Definition A.2.1. For a graph $G = (V(G), E(G))$ we say that $H = (V(H), E(H))$ is a subgraph of G if and only if for every $x \in V(H),\ x \in V(G)$ and for every $uv \in E(H),\ uv \in E(G)$.

Definition A.2.2. For a graph $G = (V(G), E(G))$ we say that $H = (V(H), E(H))$ is an induced subgraph of G if and only if H is a subgraph of G and for every $u,\ v \in V(H),\ uv \in E(H)$ iff $uv \in E(G)$.

A path is a graph whose vertices can be arranged in a linear sequence in such a way that two consecutives vertices in the sequence are adjacent. An induced path is a graph whose vertices can be arranged in a linear sequence in such a way that two vertices are adjacent if and only if they are consecutive in the sequence. A cycle is a graph whose vertices can be arranged in a cyclic sequence in such a way that two consecutives vertices in the sequence are adjacent. An induced cycle is a graph whose vertices can be arranged in a cyclic sequence in such a way

that two vertices are adjacent if and only if they are consecutive in the sequence. The length of a path is the number of vertices-1. A clique is an undirected graph where every possible edge is present. Exemple of a path, an induced path, a cycle, an induced cycle and an clique can be found in Figure A.2.

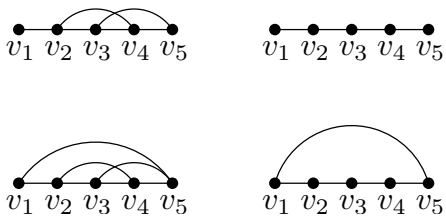


Figure A.2: On the first line: A path v_1, v_2, v_3, v_4, v_5 and an induced path v_1, v_2, v_3, v_4, v_5 . On the second line: a cycle v_1, v_2, v_3, v_4, v_5 and an induced cycle v_1, v_2, v_3, v_4, v_5.

An ordering σ of G is a bijection $\sigma : V(G) \leftrightarrow \{1, 2, ..., n\}$. Therefore for $x \in V(G)$, $\sigma(x)$ denotes the position of x in σ and for $a \in \{1, 2, ..., n\}$, $\sigma^{-1}(a)$ denotes the vertex at position a in σ. For two vertices $u,\ v$, we write that $u <_\sigma v$ iff $\sigma(u) < \sigma(v)$.

For a graph $G = (V(G), E(G))$, a subset S of $V(G)$ is called an independant set if and only if for every $u,\ v \in S$, $uv \notin E(G)$. The maximum size of an independant set of G is denoted by $\alpha(G)$. Similarly a subset S of $V(G)$ is called a clique if and only if for every $u,\ v \in S$, $u \neq v$, $uv \in E(G)$. The maximum size of a clique of G is denoted by $\omega(G)$.

A k-proper coloring of a graph $G = (V(G), E(G))$ is a partition of $V(G)$ into independant sets $S_1, \cdots, S_k$; a coloring is obtain by assigning a different color to each independant set. A graph G is k-colorable if there exists a k-proper coloring of G. The chromatic number of a graph G, denoted by $\chi(G)$, is the smallest value k such that G is k-colorable.

A.3 Orders

Definition A.3.1. A binary relation $R = (X, E)$ is said to be:

- reflexive iff for every $u \in X$, uRu.

- irreflexive iff for every $u \in X$, $u \not R u$.
- antisymmetric iff for every $u,\ v \in X$, (uRv and vRu) implies $u = v$.
- transitive iff for every $u,\ v,\ w \in X$, (uRv and vRw) implies uRw.
- total iff for every $u,\ v \in X$, uRv or vRu.

Definition A.3.2. A partial order P is an ordered pair $(X, \leq_P)$ where $\leq_P$ is a reflexive, antisymmetric and transitive binary relation over the ground set X.

A particular case of partial order is total order.

Definition A.3.3. A total order T is a partial order $(X, \leq_T)$ where the binary relation $\leq_T$ is also total.

Definition A.3.4. A strict order S is an ordered pair $(X, <_S)$ where $<_S$ is an irreflexive, antisymmetric and transitive binary relation over the ground set X.

In an order (strict or partial) P, two elements $u,\ v \in X$ are said comparable iff uRv or vRu and incomparable iff they are not comparable. For two incomparable elements $u,\ v$, we denote it by $u \parallel_R v$.

From a partial order $P = (X, \leq_P)$ we can build two relations that are often used: the strict relation that we are going to denote by $(X, <_P)$ and the covering relation that we are going to denote by $(X, \prec_P)$.

Definition A.3.5. For a partial order $P = (X, \leq_P)$, the strict order $(X, <_P)$ is define by for every $x,\ y \in X$, $x <_P y$ iff $x \neq y$ and $x \leq_P y$.

Definition A.3.6. For a partial order $P = (X, \leq)$, the covering relation denoted by $\prec_P$ over X is defined by for every $x,\ y \in X$, $x \prec_P y$ iff $x <_P y$ and $\nexists z \in X$ such that $x <_P z <_P y$.

The relation of covering captures the structure of the poset since from the covering relation we can rebuild the original poset. Indeed, from a covering relation $\prec_Q$ over X the relation $\leq_Q$ is defined by for $x, y \in X$, $x \leq_Q y$ iff $x = y$ or there exists a serie $x_0 = x,\ x_1, ..., x_j = y$ such that for $0 \leq i < j$ $x_i \prec_Q x_{i+1}$.

Definition A.3.7. The directed covering graph (also known under the name Hasse diagram) of a partial order $P = (X, \leq)$ is a representation of a partial order by its covering relation where the elements $x \in X$ are points of the plane $p(x)$ in such a way that the two following rules are respected:

- If $x <_P y$ then $p(x)$ is above $p(y)$.
- $p(x)$ and $p(y)$ are joined by a straight line iff $x \prec_P y$.

We also need the following definitions.

Definition A.3.8. Let $P = (X, \leq)$ be an ordered pair with $\leq$ an antisymmetric and transitive binary relation.

- for $u \in X$, we say that u is a source of P if and only if for every $v \in X$, $v \neq u$, $v \not\leq u$,
- for $u \in X$, we say that u is a sink of P if and only if for every $v \in X$, $v \neq u$, $u \not\leq v$,
- for two elements $u,\ v \in X$ we say that $w \in X$ is a minimal element for $u,\ v$ if and only if $w \leq u$ and $w \leq v$,
- for two elements $u,\ v \in X$ we say that $w \in X$ is a maximal element for $u,\ v$ if and only if $u \leq w$ and $v \leq w$.

We say that a binary relation (X, Q) extends a binary relation (X, P) if for every $x,\ y \in X$, xPy implies xQy. A partial order $P = (X, \leq_P)$ extends a partial order $Q = (X, \leq_Q)$ iff the binary relation $\leq_P$ extends the binary relation $\leq_Q$. In the case that P is a partial order and Q is a total order, we say that Q is a linear extension of P.

In a partial order $P = (X, \leq_P)$, a chain is a set $S \subseteq X$ such that every pair of elements is comparable, i.e. for every $u, v \in S$, $u \leq_P v$ or $v \leq_P u$. In a partial order $P = (X, \leq_P)$, an antichain is a set $S \subseteq X$ such that every pair of element is incomparable, i.e. for every $u, v \in S$, $u \neq v$, $v \parallel_P u$.

The graph induced by a partial order $P = (X, \leq)$ is the graph $G = (X, E)$ where for $x,\ y \in X$, $xy \in E$ iff $x <_P y$ or $y <_P x$. A graph is a comparability if and only if it's the induced graph of some partial order. A cocomparability graph is the complement of a comparability graph.

Definition A.3.9. A partial order $P = (X, \leq_P)$ is a lattice if and only if for every pair of elements $x,\ y \in X$ there exists an unique greatest minimal element for $x,\ y$ and an unique lowest maximal element for $x,\ y$.

For a lattice $L = (V, \leq_L)$ and two elements $x,\ y \in V$, the single greatest minimal element of $x,\ y$ is called the infimum or the meet and denoted by $x \wedge_L y$; The single lowest maximal element of $x,\ y$ is called the supremum or the join and denoted by $x \vee_L y$. A lattice $L = (V, \leq_L)$ is also denoted by $L = (V, \wedge_L, \vee_L)$.

In this thesis, we ofen talk about ordering σ or τ of the vertices of a graph G.

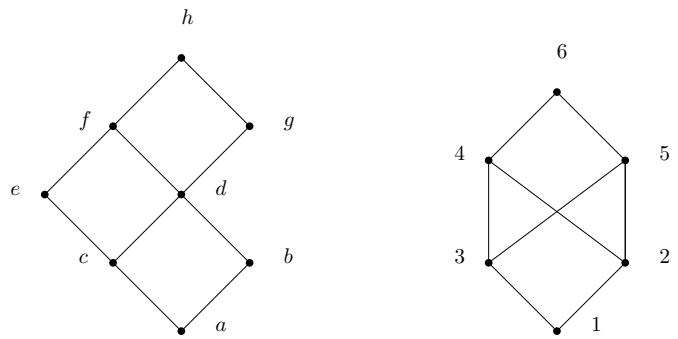


Figure A.3: On the left the directed covering graph of a lattice and on the right the directed covering graph of a poset which is not a lattice.

References

[B.88] Behrendt B. Maximal antichains in partially ordered sets. *Ars Combinatoria*, 25C:149–157, 1988. 23, 24, 25

[BCHP08] Anna Bretscher, Derek G. Corneil, Michel Habib, and Christophe Paul. A simple linear time lexbfs cograph recognition algorithm. *SIAM J. Discrete Math.*, 22(4):1277–1296, 2008. 9, 115

[BDN97] Andreas Brandstädt, Feodor F. Dragan, and Falk Nicolai. LexBFS-orderings and powers of chordal graphs. *Discrete Mathematics*, 171(1-3):27–42, 1997. 10, 100

[BPS10] Anne Berry, Romain Pogorelcnik, and Geneviève Simonet. An Introduction to Clique Minimal Separator Decomposition. *Algorithms*, 3(2):197–215, 2010. 55, 61

[CDH+] Derek G. Corneil, Jérémie Dusart, Michel Habib, Antoine Mamcarz, and Fabien de Mongolfier. A new model for graph search. *submitted at Discrete Applied Mathematics, special issue GROW conference.* 3, 94

[CDH13] Derek G. Corneil, Barnaby Dalton, and Michel Habib. LDFS-based certifying algorithm for the minimum path cover problem on cocomparability graphs. *SIAM J. Comput.*, 42(3):792–807, 2013. 2, 11, 35, 93, 102

[CDHK] Derek G. Corneil, Jérémie Dusart, Michel Habib, and E. Kőhler. On the power of graph searching for cocomparability graphs. *in preparation.* 2, 3, 35, 62

[CK] Derek G. Corneil and Ekkehard Köhler. Unpublished manuscript. 77, 91

[CK08] Derek G. Corneil and Richard Krueger. A unified view of graph searching. *SIAM J. Discrete Math.*, 22(4):1259–1276, 2008. 1, 3, 8, 9, 11, 13, 93, 94, 95, 96, 97, 98, 99, 100, 101, 106, 107, 117

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms.* The MIT Press, 2 edition, 2001. 1, 8, 9

[Cor04a] Derek G. Corneil. Lexicographic breadth first search - a survey. In *proceedings of WG*, pages 1–19, 2004. 1, 9

[Cor04b] Derek G. Corneil. A simple 3-sweep LBFS algorithm for the recognition of unit interval graphs. *Discrete Applied Mathematics*, 138(3):371–379, 2004. 9, 18, 90, 93, 96, 116

[COS09] Derek G. Corneil, Stephan Olariu, and Lorna Stewart. The LBFS structure and recognition of interval graphs. *SIAM J. Discrete Math.*, 23(4):1905–1953, 2009. 1, 3, 9, 18, 36, 75, 77, 90, 91, 93

[CT13] Christophe Crespelle and Ioan Todinca. An $o(n^2)$o(n2)-time algorithm for the minimal interval completion problem. *Theor. Comput. Sci.*, 494:75–85, 2013. 36

[DH] Jérémie Dusart and Michel Habib. A new LBFS-based algorithm for cocomparability graph recognition. *submitted at Discrete Applied Mathematics.* 3, 9, 76, 93

[DM41] Ben Dushnik and E. W. Miller. Partially ordered sets. *American Journal of Mathematics*, 63(3):pp. 600–610, 1941. 33

[DSW88] P.M. Dearing, D.R. Shier, and D.D. Warner. Maximal chordal subgraphs. *Discrete Applied Mathematics*, 20(3):181 – 190, 1988. 47

[Dus10] Jérémie Dusart. Parcours de graphes. Master's thesis, Université Paris Diderot, 2010. 102

[GH64] P.C. Gilmore and A.J. Hoffman. A characterization of comparability graphs and of interval graphs. *Canad. J. Math.*, 16:539–548, 1964. 17, 32

[Gol04] Martin Charles Golumbic. *Algorithmic Graph Theory and Perfect Graphs (Annals of Discrete Mathematics, Vol 57).* North-Holland Publishing Co., Amsterdam, The Netherlands, The Netherlands, 2004. 10, 14, 16, 55, 95, 100

[HMPR91] Michel Habib, Michel Morvan, Maurice Pouzet, and Jean-Xavier Rampon. Extensions intervallaires minimales. In *Compte Rendu à l'Académie des Sciences, présenté en septembre 91, par le Pr. G. Choquet*, pages 893–898, 1991. 36

[HMPR92] Michel Habib, Michel Morvan, Maurice Pouzet, and Jean-Xavier Rampon. Incidence structures, coding and lattice of maximal antichains. Technical Report 92-079, LIRMM, 1992. 36

[HMPV00] Michel Habib, Ross M. McConnell, Christophe Paul, and Laurent Viennot. Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. *Theor. Comput. Sci.*, 234(1-2):59–84, 2000. 11, 21, 75, 77, 82, 90, 113, 114

[HSTV05] Pinar Heggernes, Karol Suchan, Ioan Todinca, and Yngve Villanger. Minimal interval completions. In Gerth Stølting Brodal and Stefano Leonardi, editors, *ESA*, volume 3669 of *Lecture Notes in Computer Science*, pages 403–414. Springer, 2005. 36

[Kru05] Richard Krueger. *Graph Searching.* PhD thesis, Department of Computer Science, University of Toronto, 2005. 2

[KS93] Dieter Kratsch and Lorna Stewart. Domination on cocomparability graphs. *SIAM J. Discrete Math.*, 6(3):400–417, 1993. 14, 15, 116

[KSB11] Richard Krueger, Geneviève Simonet, and Anne Berry. A general label search to investigate classical graph search algorithms. *Discrete Applied Mathematics*, 159(2-3):128–142, 2011. 2, 3, 93, 94, 106, 107, 108, 109, 112

[LC62] Boland J. Lekkeikerker C. Representation of a finite graph by a set of intervals on the real line. *Fundamenta Mathematicae*, 51(1):45–64, 1962. 17

[LW13] Peng Li and Yaokun Wu. A four-sweep LBFS recognition algorithm for interval graphs. *Discrete Mathematics and Theoretical Computer Science, to appear*, 2013. 9

[Ma] T. Ma. unpublished manuscript. 77, 89, 91

[Mar75] George Markowsky. The factorization and representation of lattices. *Transactions of the American Mathematical Society*, 203:185–200, 1975. 25

[Mar92] George Markowsky. Primes, irreductibles and extremal lattices. *Order*, 9:265–290, 1992. 25

[MC12] George B. Mertzios and Derek G. Corneil. A simple polynomial algorithm for the longest path problem on cocomparability graphs. *SIAM J. Discrete Math.*, 26(3):940–963, 2012. 2, 11

[Mei05] Daniel Meister. Recognition and computation of minimal triangulations for at-free claw-free and co-comparability graphs. *Discrete Applied Mathematics*, 146(3):193–218, 2005. 21, 102

[MMNS11] Ross M. McConnell, Kurt Mehlhorn, Stephan Näherc, and Pascal Schweitzerd. Certifying algorithms. *Computer Science Review*, 5(2):119–161, 2011. 112

[MS99] Ross M. McConnell and Jeremy Spinrad. Modular decomposition and transitive orientation. *Discrete Mathematics*, 201(1-3):189–241, 1999. 21, 51, 75

[Ola91] Stephan Olariu. An optimal greedy heuristic to color interval graphs. *Information Processing Letters*, 37(1):21 – 25, 1991. 16

[Ray87] A. Raychaudhuri. On powers of interval and unit interval graphs. *Congr. Numerantium*, 59:235242, 1987. 16

[Rob71] Fred S. Roberts. On the compatibility between a graph and a simple order. *Journal of Combinatorial Theory, Series B*, 1971. 115

[RR88] G. Ramalingam and C. Pandu Rangan. A unified approach to domination problems on interval graphs. *Inf. Process. Lett.*, 27(5):271–274, 1988. 16

[RT75] Donald J. Rose and Robert Endre Tarjan. Algorithmic aspects of vertex elimination. In William C. Rounds, Nancy Martin, Jack W. Carlyle, and Michael A. Harrison, editors, *STOC*, pages 245–254. ACM, 1975. 55

[RTL76] Donald J. Rose, Robert Endre Tarjan, and George S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput.*, 5(2):266–283, 1976. 1, 9, 10, 89, 90, 100

[Sha81] Micha Sharir. A strong connectivity algorithm and its applications to data flow analysis. *Computers and Mathematics with Applications*, page 6772, 1981. 93

[Shi84] D. R. Shier. Some aspects of perfect elimination orderings in chordal graphs. *Discrete Applied Mathematics*, 7:325331, 1984. 1, 12, 13, 101

[Spi] J. Spinrad. Efficient implementation of lexicographic depth first search. submitted for publication. 12

[Spi03] Jerry Spinrad. *Efficient Graph Representations*. Fields Institute Monographs 19, American Mathematical Society, 2003. 75

[Tar72] Robert E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972. 7, 97

[Tar85] Robert Endre Tarjan. Decomposition by clique separators. *Discrete Mathematics*, 55(2):221–232, 1985. 55

[TCHP08] Marc Tedder, Derek G. Corneil, Michel Habib, and Christophe Paul. Simpler linear-time modular decomposition via recursive factorizing permutations. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP (1)*, volume 5125 of *Lecture Notes in Computer Science*, pages 634–645. Springer, 2008. 9

[TY84] Robert E. Tarjan and Mihalis Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM J. Comput.*, 13(3):566–579, 1984. 13, 101, 115

[Yan81] M. Yannakakis. Computing the minimum fill-in is np-complete. *SIAM Journal on Algebraic Discrete Methods*, 2(1):77–79, 1981. 52, 53

List of Algorithms

1 **Graph Search Paradigm (GSP).** 7
2 LBFS 10
3 LDFS 12

4 Chainclique 37
5 LocalMNS 41

6 Simplicial vertices in a cocomparability graph 59
7 Divide and conquer 61
8 List of atoms 63

9 Repeated LBFS$^+$ 76
10 Partition 81
11 Transitive orientation Algorithm 91

12 TBLS$(G, <, \tau)$ 95
13 LexUP 102
14 LexDOWN 104
15 GLS$(G, \{L, <_{GLS}, l_0, UPLAB\})$ 109
16 Computing a TBLS ordering 114