Related Orderings of AT-Free Graphs

by

Jan Gorzny

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2022

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Robert Ganian Professor, Institute of Logic and Computation, Technische Universität Wien (TU Wien)
Supervisor(s):	Jonathan Buss Professor, School of Computer Science, University of Waterloo
Internal Member:	Therese Biedl Professor, School of Computer Science, University of Waterloo
Internal Member:	Anna Lubiw Professor, School of Computer Science, University of Waterloo
Internal-External Member:	Joseph Cheriyan Professor, Department of Combinatorics & Optimization, University of Waterloo

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

Jan Gorzny was the sole author for Chapters 1, 2, 5, and 7 which were written under the supervision of Dr. Jonathan Buss. This thesis includes material from three manuscripts which have been published. Exceptions to sole authorship of material are as follows:

Research presented in Chapters 3 and 4: This research was conducted at the University of Waterloo by Jan Gorzny under the supervision of Dr. Jonathan Buss. The majority of the work was authored solely by Jan Gorzny, except Lemma 3.4.3 and Theorem 3.4.4, which was contributed by Dr. Buss. Dr. Buss also contributed intellectual input and edits to the entirety of these chapters. As a result, Dr. Buss appears on a publication related to these chapters, and will likely appear on future submissions of this work for publication.

Citations for the chapters: Gorzny and Buss [68], Gorzny [67].

Research presented in Chapter 6: This research was conducted at the University of Waterloo by Jan Gorzny under the supervision of Dr. Jonathan Buss. The majority of the work was authored solely by Jan Gorzny, except Theorem 6.4.3, which was contributed in part by Dr. Jing Huang (University of Victoria). Dr. Huang also contributed intellectual input and edits to the entirety of this chapter. As a result, Dr. Huang appears on the only publication related to this chapter, and will likely appear on any future submissions of this work for publication.

Citation for the chapter: Gorzny and Huang [70].

In all cases, my coauthors provided guidance during each step of the research and provided feedback on draft manuscripts.

Abstract

An ordering of a graph G is a bijection of $V(G)$ to $\{1, \ldots, |V(G)|\}$. In this thesis, we consider the complexity of two types of ordering problems. The first type of problem we consider aims at minimizing objective functions related to an ordering of the graph. We consider the problems Cutwidth, Imbalance, and Optimal Linear Arrangement. We also consider a problem of another type: S-End-Vertex, where S is one of the following search algorithms: breadth-first search (BFS), lexicographic breadth-first search (LBFS), depth-first search (DFS), and maximal neighbourhood search (MNS). This problem asks if a specified vertex can be the last vertex in an ordering generated by S. We show that, for each type of problem, orderings for one problem may be related to orderings for another problem of that type.

We show that there is always a cutwidth-minimal ordering where equivalence classes of true twins are grouped for any graph, where true twins are vertices with the same closed neighbourhood. This enables a fixed-parameter tractable (FPT) algorithm for Cutwidth on graphs parameterized by the edge clique cover number of the graph and a new parameter, the restricted twin cover number of the graph. The restricted twin cover number of the graph generalizes the vertex cover number of a graph, and is the smallest value $k \quad 0$ such that there is a twin cover of the graph T and $k \quad |T|$ non-trivial components of $G \quad T$.

We show that there is also always an imbalance-minimal ordering where equivalence classes of true twins are grouped for any graph. We show a polynomial time algorithm for this problem on superfragile graphs and subsets of proper interval graphs, both subsets of AT-free graphs. An *asteroidal triple* (AT) is a triple of independent vertices x, y, z such that between every pair of vertices in the triple, there is a path that does not intersect the closed neighbourhood of the third. A graph without an asteroidal triple is said to be AT-free. We also provide closed formulas for Imbalance on some small graph classes.

In the FPT setting, we improve algorithms for Imbalance parameterized by the vertex cover number of the input graph and show that the problem does not have a polynomially sized kernel for the same parameter number unless NP $\quad$ coNP/poly.

We show that Optimal Linear Arrangement also has a polynomial algorithm for superfragile graphs and an FPT algorithm with respect to the restricted twin cover number.

Finally, we consider S-End-Vertex, for BFS, LBFS, DFS, and MNS. We perform the first systematic study of the problem on bipartite permutation graphs, a subset of AT-free graphs. We show that for BFS and MNS, the problem has a polynomial time solution. We improve previous results for LBFS, obtaining a linear time algorithm. For DFS, we establish a linear time algorithm. All the results follow from the linear structure of bipartite permutation graphs.

Acknowledgements

I would like to thank everyone who helped me get to this point.

I would especially like to express my gratitude to my various advisors throughout my academic career. First, I would like to thank Professor Jonathan Buss, my supervisor, for his patience, guidance, and flexibility throughout this degree. Without his support, expert knowledge, and encouragement, I would not have finished this degree. I would also like to thank my Master's supervisors. I would like to express special thanks to Professor Jing Huang for his continued work with me after graduation and his expert introduction to an area of mathematics that made part of this thesis possible. I would also like to thank Professor Marsha Chechik for introducing me to research and the world of academia in my first graduate degree.

I would also like to thank my various co-authors and colleagues throughout my studies. I am fortunate to have had many opportunities to work with some of the smartest and hardworking academics in the world, and I am grateful to all of you. Thank you for your patience and great ideas.

To my examining committee, Professors Therese Biedl, Anna Lubiw, and Robert Ganian: thank you for your attention to my work and suggestions for future directions.

Without strong social and personal support, this degree would have been impossible. To my friends and family, thank you all for everything. Your friendship, and connections got me through the roughest times and made the best times even better. Thank you.

Table of Contents

List of Tables x

List of Figures xi

List of Algorithms xiv

1 Introduction 1

1.1 Overview of Results and Thesis Layout 7

2 Preliminaries 10

2.1 Basic Definitions 10

2.2 Definitions for Orderings 14

2.3 Graph Classes 16

2.3.1 Superfragile Graphs 18

2.3.2 Asteroidal-Triple-Free Graphs 22

2.3.3 Bipartite Permutation Graphs 23

2.4 Complexity Classes 25

2.4.1 Parameterized Complexity Classes 26

2.5 Graph Parameters 27

2.5.1 Edge Clique Cover 33

2.5.2 Restricted Twin Cover 37

2.6 Search Algorithms 46

I Type I Problems **51**

3 Cutwidth **52**

3.1 Introduction . . . 52
3.2 Preliminaries . . . 56
3.3 Small Graph Classes . . . 60
3.4 Connections to Imbalance . . . 65
3.5 A Structure of Optimal Orderings . . . 77
3.6 Edge Clique Cover Number Parameterization . . . 80
3.7 Restricted Twin Cover Number Parameterization . . . 80

4 Imbalance **92**

4.1 Introduction . . . 92
4.2 Preliminaries . . . 95
4.3 Small Graph Classes . . . 108
4.4 Structures for Optimal Orderings . . . 111
4.5 Superfragile Graphs . . . 118
4.6 Proper Interval k-Trees . . . 119
4.7 Vertex Cover Parameterizaton . . . 125
4.7.1 Kernelization Lower Bound . . . 125
4.7.2 Improved Imbalance FPT Algorithm for Graphs of Bounded Vertex Cover Number . . . 130
4.8 Remarks on Previous Attempts . . . 134
4.8.1 FPT Algorithm for Twin Cover Number . . . 134
4.8.2 Proper Interval (Bipartite) Graphs . . . 136
4.8.3 Threshold Graphs . . . 141
4.8.4 Split Graphs . . . 147

5 Optimal Linear Arrangement **148**

5.1 Introduction 148

5.2 Preliminaries 150

5.3 Superfragile Graphs 154

5.4 Restricted Twin Cover Number Parameterization 155

II Type II Problems **162**

6 End-Vertex Problem **163**

6.1 Introduction 163

6.2 Preliminaries 166

6.3 The BFS-End-Vertex Problem for Bipartite Permutation Graphs 170

6.4 The LBFS-End-Vertex Problem for Bipartite Permutation Graphs 183

6.5 The DFS-End-Vertex Problem for Bipartite Permutation Graphs 186

6.6 The MNS-End-Vertex Problem for Bipartite Permutation Graphs 193

7 Conclusions and Future Work **204**

7.1 Summary of Main Results 204

7.2 Future Work 207

References **212**

First Appearance of Symbols **226**

Index **228**

List of Tables

6.1 Known complexity results for S-End-Vertex. 165

List of Figures

1.1 An illustration of Cutwidth. 4
1.2 An illustration of Imbalance. 5
1.3 An illustration of Optimal Linear Arrangement. 6

2.1 An illustration of the layers of a graph for a specified vertex. 12
2.2 An example of twin vertices. 13
2.3 An illustration of the rank of a vertex. 15
2.4 The relationship between graph classes discussed in this work. 16
2.5 A claw graph ($K_{1,3}$). 17
2.6 A superfragile graph with an interval representation. 18
2.7 A superfragile graph. 20
2.8 A graph with an asteroidal triple. 23
2.9 Strong ordering property for bipartite graphs. 24
2.10 Illustrations of some graph parameters discussed in this work. 28
2.11 Illustrations of more graph parameters discussed in this work. 30
2.12 A graph $G = P_4$ constructed according to the operations for clique width . 31
2.13 The relationship between several parameters for graph problems. 32
2.14 An illustration of an edge clique cover of a graph G. 34
2.15 The relationship between edge clique cover number and other parameters for graph problems. 36
2.16 An illustration of the restricted twin cover parameter. 37

2.17 The relationship between restricted twin cover number and other parameters for graph problems. . . . 42
2.18 A graph and various example search orderings of it. . . . 47
2.19 The relationship between search algorithms. . . . 50

3.1 Some known complexity results for Cutwidth. . . . 54
3.2 Structure of cutwidth-minimal orderings for $K_{m;n}$ constructed by Lemma 3.3.2. . . . 61
3.3 Illustrating the preference function for rearranging vertices in a clique without increasing the imbalance of the ordering. . . . 68
3.4 An illustration of G^0 provided by as in the proof of Lemma 3.4.5. . . . 75
3.5 A graph that shows Theorem 3.5.1 is not applicable for false twins. . . . 77
3.6 An illustration of the proof of Theorem 3.5.1. . . . 78

4.1 Some known complexity results for Imbalance. . . . 94
4.2 An illustration of Lemma 4.2.13. . . . 104
4.3 An example triangulated triangle and its cutwidth-minimal ordering. . . . 109
4.4 Example Möbius ladders and their cutwidth-minimal orderings. . . . 110
4.5 An example graph has no ordering that that satisfies Theorems 4.4.1 and Theorem 4.4.2 simultaneously. . . . 112
4.6 Example proper interval k-trees along with their cutwidth-minimal orderings. 122
4.7 An edge-maximal 2-tree that is not a proper interval graph. . . . 123
4.8 The reduction of Lemma 4.7.3. . . . 126
4.9 Illustration of the sets $X(S;v)$ and $Y(S;v)$. . . . 131
4.10 A graph illustrating how the FPT algorithm for Imbalance fails for $tc(G)$. 136
4.11 A graph illustrating how to use induction for Imbalance on proper interval graphs. . . . 138
4.12 A proper interval graph with diameter 2 which does not satisfy the requirements of Lemma 3.4.2. . . . 139
4.13 A threshold graph G with the levels of a threshold partition indicated. . . . 142

4.14 A threshold graph G where $im(G) > 2cw(G)$. 144
4.15 An example of the recursive structure of threshold graphs. 145
4.16 A graph illustrating how the NP-complete reduction for Imbalance on split graphs fails. 146

5.1 Some known complexity results for Optimal Linear Arrangement. . . 149

6.1 An illustration of dominating vertices of a graph. 167
6.2 An illustration of the types of components considered in this work. 169
6.3 A vertex of a bipartite permutation graph which is not a BFS end-vertex. . 171
6.4 A vertex of a bipartite permutation graph which is a BFS end-vertex but not an LBFS end-vertex. 171
6.5 An illustration of the vertices in D_t for a vertex t. 172
6.6 An example for the proof of Lemma 6.3.3. 173
6.7 An example bipartite permutation graph with the sets used in Theorem 6.3.7 indicated. 176
6.8 An illustration of the proof of Claim 6.3.8. 178
6.9 A vertex of a bipartite permutation graph which is a DFS end-vertex. . . . 187
6.10 Case 1 of the proof of Theorem 6.5.2 illustrated. 188
6.11 An example bipartite permutation graph with the sets used in Theorem 6.6.1 indicated with substars illustrated. 195
6.12 The situation to consider for characterizing orderings generated by MNS. . 198
6.13 The situation to consider for Lemmas 6.6.7 and 6.6.8. 199

7.1 The similarities between superfragile graphs and trivially perfect graphs. . 208
7.2 The relationship between shrub depth and other parameters for graph problems. 210

List of Algorithms

2.6.1 Breadth-First Search (BFS). 47
2.6.2 Lexicographic Breadth-First Search (LBFS). 47
2.6.3 LBFS$^+$. 48
2.6.4 Depth-First Search (DFS). 49
2.6.5 Maximal Neighborhood Search (MNS). 49

6.3.1 Polynomial Time BFS-End-Vertex on Bipartite Permutation Graphs. . . 181
6.4.1 Linear Time LBFS-End-Vertex on Bipartite Permutation Graphs. 186
6.5.1 Linear Time DFS-End-Vertex on Bipartite Permutation Graphs. 191
6.6.1 Polynomial Time MNS-End-Vertex on Bipartite Permutation Graphs. . . 203

Chapter 1

Introduction

A graph is a collection of vertices along with a set of connections, called edges, between pairs of vertices. Graphs are useful for representing any number of real-world situations where there is some collection of points and connections between pairs of them; the points become the vertices of a graph while the connections naturally form the edges of the graph.

A graph, like any other collection of objects, may be arranged in multiple ways. For example, arranging a graph on a plane involves drawing the vertices of the graph as points and drawing each edge as a curve between its endpoints. Such an arrangement may be particularly suited to a graph which represents a map, where the vertices represent cities and the curves for the edges could be drawn to reflect the shape of the roads connecting cities. Not all arrangements of a graph are equal: a poorly laid out graph may not accurately scale the edges to the actual distances between cities in the above example.

Graphs can also be arranged on a line. Such an arrangement is called a *linear arrangement*, *linear layout*, or an *ordering* of a graph. Formally, an ordering of a graph G is a bijection $\quad : V(G) \ ! \ f1;:::;ng$, where $n = jV(G)j$. As in the case of arrangements on a plane, different orderings of a graph may have different properties. This kind of arrangement has a number of uses, especially when the application benefits from minimizing some property of the ordering. For example, a graph can represent a simplified model of a Very Large-Scale Integration (VLSI) circuit. As we will see later in this section, the *cutwidth* of an ordering is the maximum number of edges passing over a line between two consecutive vertices in the ordering. Raspaud et al. [122] showed that the minimal area required to layout the circuit modelled by the graph is at least the square of the cutwidth for an ordering of the graph. Thus, finding an ordering of the graph which minimizes its cutwidth may minimize the cost of the VLSI circuit. There are many more use cases for

ordering graphs so that some objective function is minimized.

Finding an ordering which minimizes an objective function is often non-trivial. In the example above, the problem is NP-complete. Without additional information or constraints, one must consider every permutation of a graph's vertices for a given objective function. Naturally, one wonders when there is an approach that is better than this naive method. By restricting the structure of the graph provided, efficient algorithms are occasionally possible. One then wonders which structure may provide the most benefit to algorithm design. Given that ordering problems enforce a linear ordering of the vertices for the solution, one wonders if graphs which somehow have a linear structure are more likely to yield efficient algorithms for these problems. This motivates one main question of the thesis: do "linear" graphs have efficient algorithms for ordering problems? It would be nice if graphs that appear linear would have straightforward optimal orderings for these problems.

Once motivated to look at "linear" graphs, one can ask other questions related to their orderings. As we are concerned about generating orderings efficiently, we can also investigate properties of orderings which are known to be efficiently generated. A *search* algorithm systematically (and efficiently) traverses the vertices of a graph one at a time until none are left. A search algorithm naturally produces an ordering of the graph, and it is therefore natural to investigate the properties of such orderings generated by graph search algorithms.

For a given graph search algorithm and graph, different orderings may be produced. Starting the search at different vertices produces different orderings, and there may be times when the search algorithm must non-deterministically choose which vertex to add to the ordering next.

We will consider the problem of determining which vertices in a graph can be the *last* vertex visited by a search algorithm. Such a vertex is called an *end-vertex*. By understanding which vertices may be last, one can understand how to build the graph in a left-to-right fashion, by starting with an initially empty graph and adding vertices according to the ordering. This is particularly useful for inductive proofs where the search algorithm is used to solve another problem. For common search algorithms, determining if a vertex is an end-vertex is NP-complete in general, but has efficient algorithms on some graphs that exhibit "linear" structure. This motivates the second main question of the thesis: can "linear" graphs be traversed efficiently so that a particular vertex is last?

This thesis will attempt to understand the answers to both of the main questions asked above. We shall call problems which are concerned with objective function minimization **Type I** problems and call problems concerned with ending a search at a particular vertex

Type II problems.

Type I Problems

We focus on three Type I problems (Cutwidth, Imbalance, and Optimal Linear Arrangement), which we now introduce. All of these problems ask if there is an ordering of the input graph which minimizes some objective function.

We start by formally defining the Cutwidth problem. Let $G = (V; E)$ be a graph and an ordering of V. The following definition is required. We will use $x < y$ for vertices $x; y \in V$ and an ordering of V if x is before (or to the left of) y, and $x > y$ if x is after (or to the right of) y in . Analogous definitions are used for and (where x may be equal to y).

The *cutwidth* after v with respect to , denoted $c(v)$, is $c(v) = |f(x; y) \in E \mid x$ v and $v < y|$. The *cutwidth* of is $cw( ) = \max_{v \in V} fc(v)g$. The *cutwidth* of G, denoted $cw(G)$, is the minimum of $cw( )$ over all orderings of V.

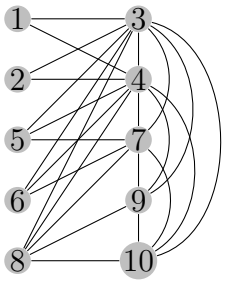
Problem 1.0.1 (Cutwidth). *Given a graph G and a positive integer k, determine if $cw(G)$ k.*

Ideally, a solution to Cutwidth finds an ordering of G such that $cw( )$ k. An example of a Cutwidth ordering is shown in Figure 1.1.

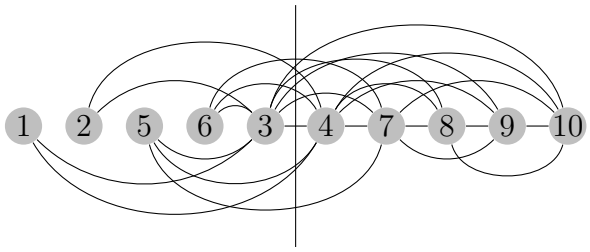
Cutwidth was chosen for study due to its status on some interesting graph classes. There are no known efficient algorithms on so-called interval graphs, and determining if there are any has been an open question for at least ten years. Interval graphs are a subclass of so-called asteroidal-triple-free (AT-free) graphs, which are considered to exhibit linear structure. These graph classes are formally defined in Section 2.3, while a complete picture of the complexity results along with applications for this problem can be found in Section 3.1. On other sub-classes of AT-free graphs, like the so-called threshold graphs, there are efficient algorithms for this problem.

The existence of efficient algorithms for Cutwidth on threshold graphs motivated the investigation of related ordering problems. Lokshtanov et al. [104] showed that another ordering problem, Imbalance, has a solution that is at least twice the cutwidth of a graph. As a result, one wonders if the algorithms for Cutwidth may help or inspire those for Imbalance, which we now define.

Informally, Imbalance asks for an ordering which minimizes the sum of the *imbalance* of vertices, where the imbalance of a vertex is the difference of sizes in its neighbourhood to its left and to its right in the ordering.



(a) A graph G.



$c\ (3) = 11$

(b) An ordering of G such that $cw(\ ) = 11$. The vertical line crosses 11 edges, the most of any line between two vertices in the ordering. By trying all possible permutations of $V(G)$, one can check that $cw(G) = 11$, and thus this ordering is cutwidth-minimal.

Figure 1.1: An illustration of Cutwidth.

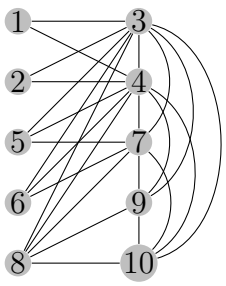
Formally, let $G = (V; E)$ be a graph and an ordering of V. For $v \in V$, pred (v) and succ (v) respectively denote the number of neighbours of v that precede (respectively succeed) v in an ordering . The *imbalance* of v with respect to , denoted (v), is $|\text{succ}\ (v) - \text{pred}\ (v)|$. The *imbalance* of is $im(\ ) = \sum_{v \in V} (v)$. The *imbalance* of G, denoted $im(G)$, is the minimum of $im(\ )$ over all orderings of V.

Problem 1.0.2 (Imbalance)**.** *Given a graph G and a positive integer k, determine if $im(G) \leq k$.*

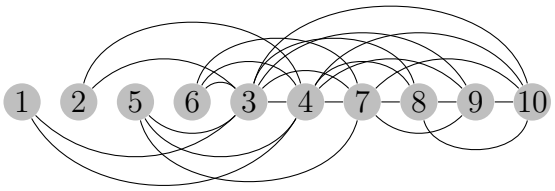
As in the case of Cutwidth, a solution to Imbalance ideally also finds an ordering of G such that $im(\ ) \leq k$. An example of an Imbalance ordering is shown in Figure 1.2.

In this work, we first investigate the connection between these two problems. We show that the complexity of Imbalance matches that of Cutwidth on various restricted graph classes, and that graphs have optimal orderings for both which are similar in structure.

We then consider another problem related to Cutwidth: Optimal Linear Arrangement (OLA). Informally, the Optimal Linear Arrangement problem asks for an ordering which minimizes the sum of weights of edges in an ordering, where the weight of an edge is one more than the number of vertices between its endpoints in the ordering. This is equivalent to minimizing the sum of edges passing across every two consecutive vertices in the ordering. Thus, there is at least a superficial connection between Optimal Linear Arrangement and Cutwidth: the former asks to minimize the sum of



(a) A graph G.



(v) 2 2 3 3 1 1 1 1 3 5

(b) An ordering of G such that $im(\ ) = 22$. By trying all possible permutations of $V(G)$, one can check that $im(G) = 22$, and thus this ordering is imbalance-minimal.

Figure 1.2: An illustration of Imbalance.

cuts, while the latter asks to minimize the maximum cut. That is, Optimal Linear Arrangement asks to minimize $W(\ ) = \sum_{v \in V} c\ (v)$ instead of the maximum value of $c\ (v)$ over all $v \in V$ (see, e.g., Díaz et al. [44]).

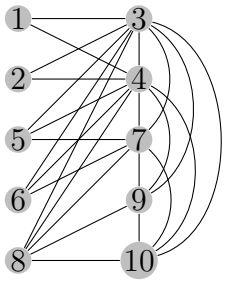
Formally, given an ordering of the vertices of a graph $G = (V; E)$, the *weight* of an edge $(u; v) \in E$ is $j\ (u) \quad (v)j$ and is denoted $w\ (e)$. The *weight* of an ordering is $W(\ ) = \sum_{(u;v) \in E} w\ (e)$. An *optimal linear arrangement* of G is a layout with the minimum weight, i.e., $\arg\min\ W(\ )$. The *minimum weight* of G is $W(G) = \min\ W(\ )$.

Problem 1.0.3 (Optimal Linear Arrangement)**.** *Given a graph G and a positive integer k, determine if $W(G)$ k.*

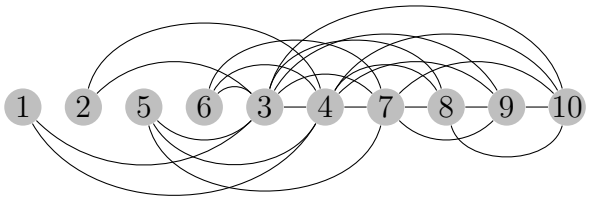
Ideally, a solution to Optimal Linear Arrangement finds an ordering of G such that $W(\ )$ k. An example of an Optimal Linear Arrangement ordering is shown in Figure 1.3. The Optimal Linear Arrangement problem has also been called the Minimum Linear Arrangement problem (Even [48]).

It turns out that Optimal Linear Arrangement solutions often look like those for Cutwidth and Imbalance. We are able to show that for Cutwidth and Imbalance, there are always orderings that group true twins, vertices with the same closed neighbourhood, together in the ordering, matching a previously established result of Fellows et al. [49] for Optimal Linear Arrangement. Many techniques for one problem also inspire results for another.

However, we do not have a complete picture of how these problems relate to each other, and the fact that these problems have solutions which are similar has not been previously noted. We are hopeful that these theorems describing optimal orderings for these problems



(a) A graph G.



e	$w(e)$	e	$w(e)$	e	$w(e)$
$(1;3)$	4	$(1;4)$	5	$(2;3)$	3
$(2;4)$	4	$(3;4)$	1	$(3;5)$	2
$(3;6)$	1	$(3;7)$	2	$(3;8)$	3
$(3;9)$	4	$(3;10)$	5	$(4;5)$	3
$(4;6)$	2	$(4;7)$	1	$(4;8)$	2
$(4;9)$	3	$(4;10)$	4	$(5;7)$	4
$(6;7)$	3	$(7;8)$	1	$(7;9)$	2
$(7;10)$	3	$(8;9)$	1	$(8;10)$	2
$(9;10)$	1				

(b) An ordering of G such that $W(\ ) = 66$. By trying all possible permutations of $V(G)$, one can check that $W(G) = 66$, and thus this ordering has minimum weight for G.

Figure 1.3: An illustration of Optimal Linear Arrangement.

are useful for understanding these problems and future results. This thesis takes a first step towards considering all of these problems at once, taking results from one problem to apply them to another. Specific contributions and results are summarized in later chapters, when the relevant graph classes have been defined. A list of contributions of the thesis for these problems is found in the conclusion (Chapter 7).

Type II Problems

A graph search *visits* every vertex in a (connected) graph and naturally produces an ordering of the graph. In this work, we consider the question of whether a particular vertex can be visited last for several search algorithms; this is the *end-vertex* problem, denoted S-End-Vertex for a search algorithm S. In this thesis, we discuss S-End-Vertex for various search algorithms S, as our family of Type II problems.

Formally, call the last visited vertex of a search S of a graph an *end-vertex* of S of the

graph. We now formally define the S-End-Vertex for a search algorithm S:

Problem 1.0.4 (S-End-Vertex). *Given a graph $G = (V; E)$, a vertex t 2 V, and a search algorithm S, determine if there exists an ordering $: V ! f1;:::;ng$ (where $n = jVj$) generated by S of V such that $(t) = n$.*

A related problem, the S-Beginning-End-Vertex problem asks if it is possible to end a search S at a vertex t after starting at vertex s. Many results for S-End-Vertex carry over to S-Beginning-End-Vertex by considering all possible starting vertices. The property of being last in a search is particularly useful in inductive proofs for algorithms which employ such searches. For these reasons, we focus only on the S-End-Vertex problem, and perform the first systematic study of the problem on a single graph class (so-called bipartite permutation graphs, which are defined in Section 2.3.3).

1.1 Overview of Results and Thesis Layout

We obtain several results for problems of both types. Our focus is on the complexity of these problems in the both the "classical" setting and "parameterized" setting. In the "classical" setting, we study algorithms for these problems where we measure the run time solely in terms of the size of the input graph (number of vertices and edges). In the "parameterized" setting, we study algorithms for these problems where the run time is measured in the size of the input graph and another input parameter. Along the way, we prove some structural results for these problems which may be of general interest and establish connections between related problems.

In the classical complexity setting for Type I problems, we obtain polynomial time algorithms on some subsets of AT-free graphs. For Imbalance, we use the problem's similarities to Cutwidth to obtain an $O(n^2)$ (where $n = jV(G)j$) time algorithm on the so-called superfragile graphs (defined in Section 2.3). In turn, we use the similarity of Optimal Linear Arrangement to Imbalance to obtain an $O(n^2)$ time algorithm on superfragile graphs for Optimal Linear Arrangement as well. We also show that for some of the so-called proper interval graphs (also defined in Section 2.3), Imbalance has an algorithm that runs in time $O(n)$. Complexity results for general AT-free graphs remain out of reach, as these problems remain unsolved for other subsets of AT-free graphs, as explained in Section 4.8.

In order to obtain these algorithms, we study the equivalence classes of *true twins*, vertices with the same closed neighbourhood, in graphs. From this investigation, we are

able to show that for both Cutwidth and Imbalance, there are optimal orderings for each of these problems where equivalence classes of true twins appear consecutively for *any* graph. These results match an existing result for Optimal Linear Arrangement, and establish additional similarities between these problems. Exploring those similarities more, we are able to establish some conditions for when an cutwidth-minimal ordering is also imbalance-minimal. In turn, we obtain formulas for the imbalance of several small classes of graphs: complete (bipartite) graphs, Möbius ladders, and triangulated triangles.

In the parameterized complexity setting for Type I problems, we focus on exploring the behaviour of true twins in these problems. Using the theorems which state that these problems have optimal orderings where the classes of true twins are consecutive, we are able to establish several fixed-parameter tractable (FPT) algorithms. First, we show that Cutwidth has an FPT algorithm when the parameter is the so-called edge clique cover number of the graph (defined in Section 2.5). Aiming to generalize FPT results of Fellows et al. [52] and Lokshtanov [103] for Type I problems when the parameter is the so-called vertex cover number of the graph (defined in Section 2.5), we considered the applicability of these theorems to related parameters. We first attempted to use them for the so-called twin cover number of the graph (defined in Section 2.5), but we were unsuccessful (see Section 4.8.1 for an explanation). Instead we define a *new* parameter in Section 2.5.2, the *restricted twin cover number* of a graph, which overcomes the problems with the general twin cover case. As a result, we obtain FPT algorithms for Cutwidth and Optimal Linear Arrangement when the parameter is the restricted twin cover number of the graph. For Imbalance, such a result is not novel (it is implied by another algorithm, as explained in Section 4.1), but we once again use the similarities between Cutwidth and Imbalance to obtain some FPT results. We show an improved FPT algorithm for Imbalance when the parameter is the vertex cover number of graph, and that Imbalance has no polynomial sized kernel (defined in Section 2.4) with the same parameter.

We study the Type II problems only in the setting of classical complexity. We make conjectures about the parameterized complexity of these problems in the conclusion (Section 7.2), but focus on exploring how the linearity of AT-free graphs helps these problems in the context of bipartite graphs. We establish S-End-Vertex results for Breadth-first Search, Lexicographic Breadth-First Search, Depth-First Search, and Maximal Neighbourhood Search (all defined in Section 2.6). We show that for the search algorithms Lexicographic Breadth-First Search and Depth-First Search, the end-vertex problem has a linear time solution on AT-free bipartite graphs. For Breadth-First Search and Maximal Neighbourhood Search, the end-vertex problem has polynomial time algorithms on AT-free bipartite graphs.

Thesis Layout

The layout of this thesis is as follows. Chapter 2 provides the relevant definitions and notions used throughout the thesis. Each problem is discussed in its own chapter which includes background and previous results for the problem. Chapter 3 discusses results for Cutwidth which closes some gaps in the literature, and includes a new structural theorem. Chapter 4 demonstrates that some results for Cutwidth apply to Imbalance. Chapter 5 takes the first steps in showing that results for Imbalance may apply to Optimal Linear Arrangement. The S-End-Vertex problem is discussed in Chapter 6. The thesis concludes with Chapter 7, which summarizes key results and lists open problems and directions for future research.

Chapter 2

Preliminaries

2.1 Basic Definitions

All graphs in this work are finite, undirected, and without multiple edges or loops unless otherwise specified. For a graph $G = (V; E)$, we will denote $n = |V|$ and $m = |E|$. The following definitions are standard.

If $X \subseteq V$, then we define the subgraph *induced* by X to be $G[X] = (X; E_X)$ where $E_X = \{(u; v) \in E \mid u \in X \text{ and } v \in X\}$. We say that G contains an induced subgraph H if there is a set of vertices $X \subseteq V(G)$ such that $G[X] = H$.

A *complete graph* (or *clique*) is a graph whose vertices are pairwise adjacent. We will use K_n to denote a complete graph on n vertices. A complete graph or clique is *maximal* if it cannot be extended by including one more vertex which is adjacent to every vertex in the clique.

An *independent set* is a set $I \subseteq V$ of vertices with no edges among them (i.e., $(I \times I) \setminus E = \emptyset$). We will use I_n to denote an independent set on n vertices.

A *cycle* in a graph is a sequence of vertices $\{u_0; \ldots; u_{t-1}\}$ such that $(u_i; u_{i+1})$ is an edge for all $0 \leq i < t - 1$, $(u_0; u_{t-1})$ is an edge, and all vertices are unique. A cycle $\{u_0; \ldots; u_{t-1}\}$ has a *chord* if there is an edge $(u_i; u_j) \in E \setminus \{(u_0; u_{t-1})\}$ where $j > i$ and $j - i > 1$; a cycle is *chordless* if it has no chords. The *length of a cycle* on t vertices is t.

A *path* in a graph is a sequence of vertices $\{u_0, \ldots, u_{t-1}\}$ such that (u_i, u_{i+1}) is an edge for all $0 \leq i < t - 1$ and all vertices are unique. A path $\{u_0, \ldots, u_{t-1}\}$ has a chord if there is an edge $(u_i, u_j) \in E$ and where $j > i$ and $j - i > 1$; a path is chordless if it has no chords. We will use P_t to denote a chordless path on t vertices. For any path P_t, the vertices u_0 and u_{t-1} are the *endpoints* of the path. A (u, v)-path is a path between u and v where u and v are the endpoints. If t is odd, then $u_{\lfloor t/2 \rfloor}$ is the *midpoint* of the path, while if t is even, then the vertices $u_{t/2-1}$ and $u_{t/2}$ are the *midpoints* of the path. The *length of a path* on t vertices is $t - 1$.

A graph G is *connected* if there is a (u, v)-path for every $u, v \in V(G)$.

A *component* of a graph is any of its maximal connected subgraphs. A component is *trivial* if it contains no edges. A component is *non-trivial* if it contains at least one edge (and therefore at least two vertices).

The *open neighbourhood* of a vertex $v \in V(G)$, denoted $N_G(v)$, is the set $\{u \in V \mid (v, u) \in E\}$ of vertices adjacent to v. The *closed neighbourhood* of a vertex, denoted $N_G[v]$, is the open neighbourhood of the vertex along with the vertex itself, i.e., $N_G[v] = N(v) \cup \{v\}$. In both cases, we will drop the subscript when it is clear from context.

If $X \subseteq V$, then the *open neighbourhood of a set* X, denoted $N(X)$, is $\bigcup_{v \in X} N(v) \setminus X$. The *closed neighbourhood of a set* X, denoted $N[X]$, is $\bigcup_{v \in X} N[v]$.

The *degree* of a vertex v is the size of its open neighborhood, i.e., $|N(v)|$, and is denoted $d_G(v)$, though we will drop the subscript when it is clear from context.

A vertex v is *isolated* if $d_G(v) = 0$.

A vertex v is a *pendant* if $d_G(v) = 1$.

The *maximum degree* of a graph G, denoted $\Delta(G)$, is $\max_{v \in V(G)} d_G(v)$.

A vertex v is *simplicial* if $N(v)$ induces a complete graph. For a simplicial vertex v, $N[v]$ also induces a complete graph.

For a connected graph G, a set of vertices $X \subseteq V$ is a *separator* if $G(V \setminus X)$ is disconnected. Such a set X is a *clique separator* if it also induces a complete graph. A set of vertices $X \subseteq V$ is an *(A, B)-separator* if $G(V \setminus X)$ is disconnected and the disjoint sets A and B are in two separate components of $G(V \setminus X)$.

A vertex $v \in V(G)$ is a *cut vertex* if $G \setminus v$ has more components than G.

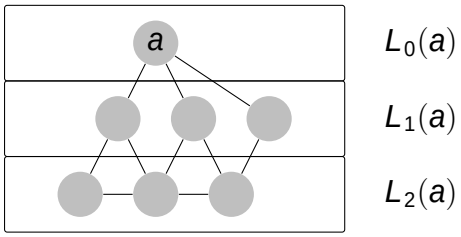


Figure 2.1: An illustration of the layers of a graph for the vertex a. The rectangles show the layers $L_i(a)$ for $0 \quad i \quad \text{ecc}_G(a)$. Since $\text{ecc}_G(a) = 2$, the set $L_2(a)$ is also the set of eccentric vertices of a.

For a subset $S \quad V$, a vertex $v \; 2 \; S$ is *universal to* S if $S \quad N(v)$. We will use $U(S)$ to denote the set of universal vertices for $S \quad V$, i.e., $U(S) = fv \; j \; v \; 2 \; S$ and $S \quad N[v]g$.

An $(x;y)$-path P *misses* (or *avoids*) z if $V(P) \setminus N[z] = ;$, that is, P contains neither z nor a neighbour of z; otherwise the vertex z is said to *intercept* (or *hit*) the path P.

The *distance* between two vertices u and v in a graph G, denoted by $d_G(u;v)$, is the length of a shortest path between u and v in G. We will drop the subscript when it is clear from context.

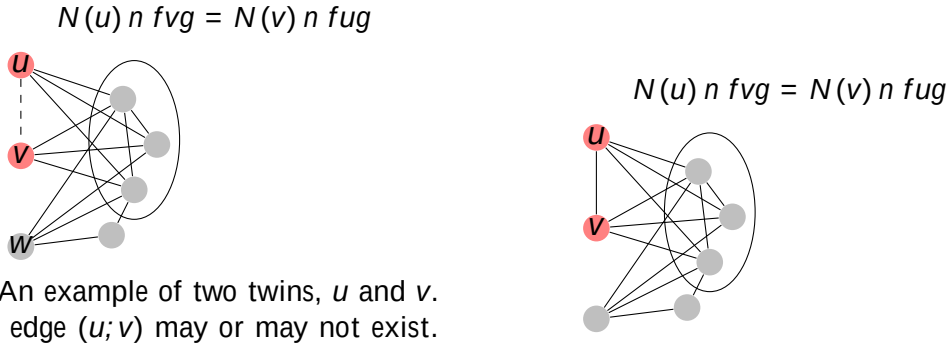
The *diameter* of a graph G, denoted $\text{diam}(G)$, is the maximum distance between any two vertices in G, i.e., $\text{diam}(G) = \max_{u;v2V(G)} d_G(u;v)$. Two vertices $u;v$ are said to be *diametrical* if $d_G(u;v) = \text{diam}(G)$.

For a vertex w, we use $L_`(w)$ to denote the set of all vertices $u$ with $d_G(u;w) = `$. The set $L_i(w)$ is called the ith *layer* of the graph with respect to w. The maximum value $`$ for which $L_`(w) \neq ;$ is called the *eccentricity* of w and is denoted by $\text{ecc}_G(w)$. The subscript will be dropped when it is clear from context. When $` = \text{ecc}(w)$, the vertices of $L_`(w)$ are called *eccentric vertices* of w. These concepts are illustrated in Figure 2.1.

For $v \; 2 \; V$, we will occasionally use $E(v)$ denote the edges of G that are incident with v. Note that $jE(v)j = d_G(v) = jN(v)j$.

For disjoint sets $X;Y \quad V$, we will use $E(X;Y)$ denote the edges of G that have one endpoint in X and the other in Y, i.e., $E(X;Y) = f(x;y) \; 2 \; E \; j \; x \; 2 \; X$ and $y \; 2 \; Y g$.

To *subdivide an edge* $e = (u;v)$ is to delete e and add a new vertex x and join x to u and v.



(a) An example of two twins, u and v. The edge $(u;v)$ may or may not exist. The vertex w is not a twin of either u or v since it has a neighbour outside the common neighbourhood of u and v, which is in the oval.

(b) An example of two true twins, u and v. The edge $(u;v)$ must exist.

Figure 2.2: An example of twin vertices.

Two vertices u and v are *twins* if they have the same neighbours, except possibly for each other; that is, $N(u) \setminus \{v\} = N(v) \setminus \{u\}$. Equivalently, u and v are twins if each $w \in V \setminus \{u;v\}$ satisfies $(w;u) \in E$ if and only if $(w;v) \in E$. Two twins u and v are *true twins* if they have an edge between them; i.e., $N[u] = N[v]$. An illustration of twins is provided in Figure 2.2. Twins which are not true twins, i.e., twins u and v such that $(u;v) \notin E$, may be referred to as *false twins*. A set of vertices $Y \subseteq V$ is a *set of true twins* if for every pair of vertices $u;v \in Y$, u and v are true twins. Inclusion-wise maximal sets of true twins in a graph form equivalence classes of the vertices of a graph (Ganian [58]). An equivalence class for the relation of "being true twins" may consist of a single vertex.

We emphasize the following definition regarding twins:

> A set $X \subseteq V(G)$ is a *set of twins* if, either statement holds (i) for any $u;v \in X$, u and v are false twins, or (ii) for any $u;v \in X$, u and v are true twins. That is, we will only use the phrase "set of twins" if the set consists entirely of true twins or entirely of false twins.

The next observation is also important for dealing with the set of universal vertices of a graph.

Observation 2.1.1. *For any graph G, the set $U(G)$ is an equivalence class of true twins and a clique.*

2.2 Definitions for Orderings

An *ordering* (or a *(linear) layout* or a *linear arrangement*) of (a subset of) the vertices of a graph is a sequence $\langle v_1, v_2, \ldots, v_k \rangle$, with each v_i distinct. We shall freely use set operations and notation on orderings, and also the following.

(v) denotes the index of the vertex v in .

(i) denotes the ith vertex in , for 1 i j j.

denotes the concatenation of (disjoint) orderings and .

The relation $<$ is defined by $u < v$ if and only if u precedes v in . Relations $>$, and are defined analogously. We extend these to sets of vertices: e.g., $x < \{y, z\}$ if and only if $x < y$ and $x < z$. More generally, for two sets X and Y, we say that $X < Y$ if and only if $x < Y$ for all $x \in X$.

For an element x of , $<_x$ denotes the ordering induced by on the set $\{y \in V \mid y < x\}$. The orderings $_x$, $>_x$, and $_x$ are defined analogously. More generally, for a set $X \subseteq V$, $_X$ denotes the ordering imposed by on X.

We may use $<_i$ for 1 i j j to refer to $<_{(i)}$. The symbols $>_i$, $_i$, and $_i$ are defined analogously.

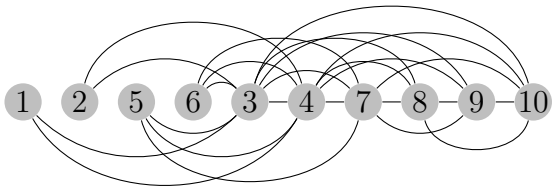
For an ordering , ^{R} denotes the reverse ordering of .

A set X V is *consecutive* in an ordering of V if there exists an index 1 i j j and an index $i < j$ j j such that j $i = |X|$ and for all i k j, $(k) \in X$.

The notation $<_X$, where X is a consecutive set in , denotes the ordering of restricted to the vertices prior to the consecutive set X. Similarly, the notation $_X$, where X is a consecutive set in , denotes the ordering of restricted to the vertices prior to the consecutive set X and the vertices of X. We will use analogous for $>$ and .

The following definitions are related to the Type I problems that we study.

Recall that the *cutwidth* after v with respect to , denoted c (v), is c $(v) = |\{(x, y) \in E \mid x$ v and $v <$ $y\}|$. For convenience, we may occasionally write c (i) instead of c ((i)) to represent the cutwidth after the ith vertex of .



rank (v)	2	2	3	3	1	1	1	1	3	5
(v)	2	2	3	3	1	1	1	1	3	5

Figure 2.3: An illustration of the rank of a vertex.

Given a vertex $v \in V$, pred (v) and succ (v) respectively denote the number of neighbours of v that *precede* (respectively *succeed*) v in an ordering π. That is, pred $(v) = |\{u : u <_\pi v\} \cap N(v)|$ and succ $(v) = |\{u : u >_\pi v\} \cap N(v)|$. The *imbalance* of v with respect to an ordering π, denoted $\pi(v)$, is $|\text{succ } (v) - \text{pred } (v)|$.

Given a set $X \subseteq V$ and an ordering π of V, define $\pi(X) = \sum_{v \in X} \pi(v)$.

Given a set of twins $Y \subseteq V$ and an ordering π of Y where the vertices of Y appear consecutively, define pred $(Y) =$ pred (y) where $y \in Y$ is leftmost in π. Similarly, given a set of twins $Y \subseteq V$ and an ordering π of Y where the vertices of Y appear consecutively, define succ $(Y) =$ succ (y) where $y \in Y$ is rightmost in π.

A vertex v is *perfectly balanced* in an ordering π if $d(v)$ is even and $\pi(v) = 0$ or if $d(v)$ is odd and $\pi(v) = 1$.

The *rank*[1] of a vertex v_i with respect to an ordering π is denoted by rank (v_i) and is equal to succ (v_i) − pred (v_i). Note that $|\text{rank } (v_i)| = \pi(v_i)$. The rank of a vertex is illustrated in Figure 2.3.

In many cases, we will need to rearrange vertices in an ordering π. Suppose that π is written

$$\pi = (1)\ (2) \ldots (i-1)\ (i)\ (i+1) \ldots (n-1)\ (n).$$

We say that we move $v = \pi(i)$ *backward* (or *to the left*) to a position j where $0 < j < i$ when we replace π by the ordering

$$\pi' = (1)\ (2) \ldots (j-1)\ (i)\ (j) \ldots (i-1)\ (i+1) \ldots (n-1)\ (n).$$

[1] Some authors call this the *force* of a vertex in the context of Optimal Linear Arrangement, e.g., Fellows et al. [49].

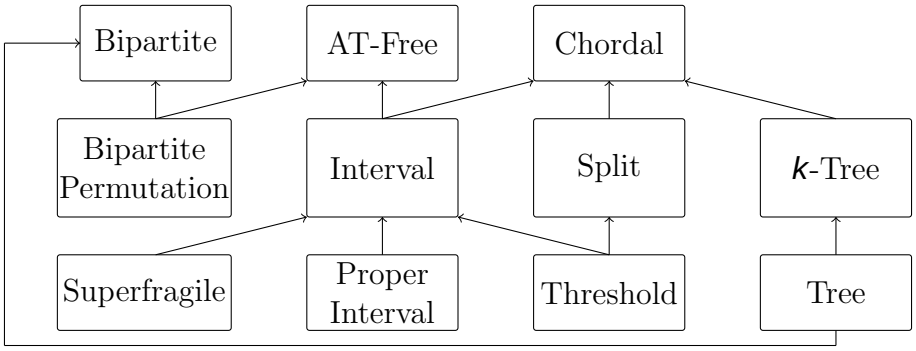


Figure 2.4: The relationship between graph classes discussed in this work. An arrow from class A to class B indicates that class A is contained within class B.

Similarly, we say that we move $v =$ (i) *forward* (or *to the right*) to a position j where n $j > i$ when we replace by the ordering

$$^{0} = (1)\ (2) ::: (i \quad 1)\ (i+1) ::: (j \quad 1)\ (i)\ (j)\ (n \quad 1)\ (n):$$

2.3 Graph Classes

We shall consider various classes of graphs in this work, which we now define. The relationship between these graph classes is shown in Figure 2.4.

A graph G is *bipartite* if V can be partitioned into two sets X and Y such that all edges have one endpoint in X and the other in Y; each set X and Y is called a *partite* set of G. We will often write $G = (X; Y; E)$ for a bipartite graph, where X and Y are the partite sets of G. A *bigraph* is a bipartite graph. A *complete bipartite graph*, denoted $K_{r;s}$, is a bipartite graph on $r + s$ vertices such that there are r vertices in one partite set and s in the other, and moreover, every vertex in one partite set is adjacent to every vertex in the other partite set. The *claw* graph is $K_{1;3}$ (shown in Figure 2.5) and will be used in some definitions and proofs.

A *chordal* (or *triangulated*) graph is a graph with no induced cycle of size at least four. We consider the following sub-classes of chordal graphs.

A *tree* is a connected acyclic graph, i.e., one which does not contain any cycles. A *star* is a tree where all vertices except one have degree 1.

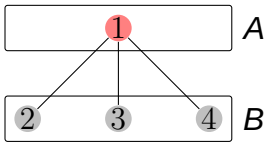


Figure 2.5: A claw graph ($K_{1,3}$) with each partite set indicated.

An *interval* graph is one in which each vertex v may be identified with an interval I_v of the real line, such that $(u,v) \in E$ if and only if $I_u \cap I_v \neq \emptyset$ (i.e., two vertices are adjacent if and only if their respective intervals intersect). An example is shown in Figure 2.6. All interval graphs are chordal. Lekkerkerker and Boland [96] showed that a chordal graph is an interval graph if and only if it contains no so-called asteroidal triple (defined in Section 2.3.2).

An interval graph is *proper* if the intervals can be chosen so that no interval contains another; i.e., $I_u \not\subseteq I_v$ for every $u \neq v$. An interval graph is proper if and only if it contains no claw (see Figure 2.5). Proper interval graphs are precisely the *unit interval* graphs: interval graphs whose representations can be drawn with unit length intervals. Golumbic [65] provides a proof for both of these facts.

A *split* graph is a graph whose vertex set can be partitioned into a clique and an independent set. Such a partition is called a *split partition*. All split graphs are chordal (Hammer and Földes [72]).

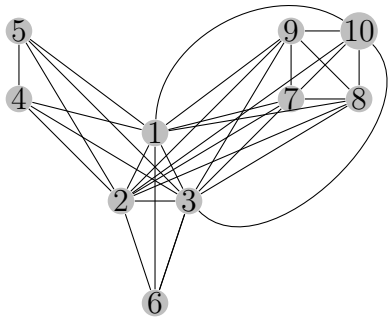
A split graph is a *threshold graph* if and only if it has a split partition (C, I) such that vertices of I (and equivalently the vertices of C) can be ordered by neighbourhood inclusion. Such a split partition is called a *threshold partition*, and is defined in Section 4.8.3.

A *k-tree* is a graph that can be obtained by starting with a $k+1$ clique and repeatedly adding vertices so that each new vertex added has precisely k neighbours which form a clique. All k-trees are chordal and have no clique of size greater than $k+1$ (see, e.g., Brandstädt et al. [18] and Patil [118]).

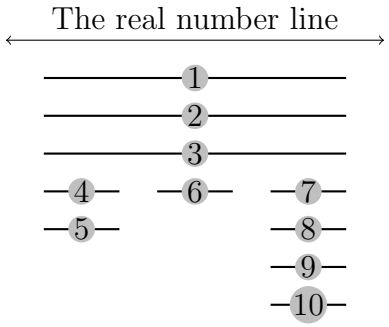
A graph G is *superfragile* if G can be constructed with the following two operations **O1** and **O2**:

O1 Adding a clique U to a union of disjoint cliques $C_1, \ldots, C_k$, and adding all edges between vertices of U and C_i for every $1 \leq i \leq k$.

O2 Taking the union of disjoint superfragile graphs.



(a) A superfragile graph G. Observe that every vertex in $U(G) = \{1, 2, 3\}$ has an edge to every vertex of the disjoint cliques $K_2 = \{4, 5\}$, $K_1 = \{6\}$, and $K_4 = \{7, 8, 9, 10\}$.



(b) An interval representation of G. Each thick line represents an interval on the real number line corresponding to the vertex of the indicated label. This representation is not unique.

Figure 2.6: A superfragile graph (left) with an interval representation (right). Since the graph has an interval representation, it is also an interval graph.

From this definition, it is clear that superfragile graphs contain no induced cycles of length greater than 3 or so-called asteroidal triples (defined in Section 2.3.2). Therefore, superfragile graphs are interval graphs and also chordal graphs.

An alternative definition provided by Preissmann et al. [119] is proved to be equivalent in Section 2.3.1 below, which also remarks on the naming of this class of graphs.

2.3.1 Superfragile Graphs

Lilleeng [99] defines superfragile graphs as follows: a graph G is *superfragile* if G can be constructed with operations **O1** and **O2**.

The operation of taking two graphs G_1 and G_2 and adding edges between every vertex of G_1 and every vertex of G_2 is called a *complete join* of two graphs. We may use this terminology when constructing superfragile graphs throughout the thesis.

We make the following observation about superfragile graphs constructed using only operation **O1**, i.e., superfragile graphs which are connected. The observation will be useful in Sections 4.5 and 5.3.

Observation 2.3.1. *Let G be a connected superfragile graph which is not a complete graph. Then, G was constructed using only operation* **O1** *by a complete join of U and the graph consisting of the disjoint cliques $C_1, \ldots, C_k$. The $k+1$ equivalence classes of true twins in G are the sets $C_1, \ldots, C_k$ along with the set U. Moreover, $U(G) = U$.*

We now turn to another definition of superfragile graphs. After introducing it, we will show the equivalence of the definitions. Preissmann et al. [119] define superfragile graphs as follows. Consider the following two rules **RA** and **RB** which can be used to eliminate vertices from a graph:

RA If there is no induced P_3 having node x as an endpoint, then x and all incident edges are removed.

RB If there is no induced P_3 having node x as a midpoint, then x and all incident edges are removed.

An *elimination ordering* of a graph G is an ordering describing the order to remove vertices of a graph (usually to satisfy some special property). A graph is superfragile if there is an elimination ordering using the rules **RA** and **RB** such that at every stage of the elimination ordering, every remaining vertex is eligible to be removed.

We briefly remark on the naming of this class of graphs. Chvátal called graphs *brittle* if rules similar to **RA** and **RB**, using P_4 instead of P_3, could be used to obtain an elimination ordering (see Preissmann et al. [119]). Preissmann et al. [119] considered graphs where every vertex was eligible for elimination during every stage, and called these graphs *superbrittle*. The name "superfragile" arises from the similarity to superbrittle graphs.

The following proposition proves the equivalence of the two definitions above, which was not explicitly shown by Lilleeng [99]. Figure 2.7 shows how the elimination rules above may be used for a superfragile graph.

Proposition 2.3.2. *Graphs which can be constructed using operations* **O1** *and* **O2** *are exactly the graphs which have an elimination ordering using rules* **RA** *and* **RB**, *where every vertex in the graph is eligible to be removed at every stage of the elimination ordering.*

Proof. Suppose first that G was constructed according to operations **O1** and **O2**. Consider any vertex v and let component C be the component of G containing v. If C is a clique, then C contains no induced P_3, in which case no vertex of C is the endpoint of any P_3 and we can apply rule **RA** to any vertex in the component at any point during the elimination

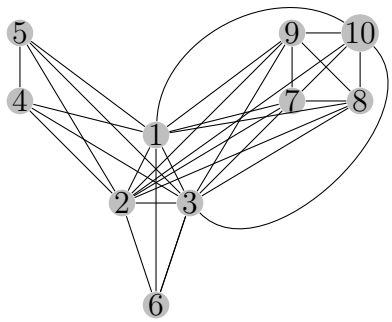


Figure 2.7: A superfragile graph G. Observe that $U(G) = \{1, 2, 3\}$. Note that rule **RA** is always able to remove vertices of $U(G)$, while rule **RB** is always able to remove any vertex of $V(G) \setminus U(G)$. Either rule is applicable once G is a clique.

ordering. If C is not a clique, then it must be that C was constructed by joining all vertices of $U(C)$, the universal vertices of the component, to some cliques $C_1, \ldots, C_k$. In this case, we consider cases based on whether $v \in U(C)$.

Case 1: $v \in U(C)$. We claim that there is no induced P_3 having v as an endpoint. Suppose to the contrary that v is the endpoint of some induced $P_3 = \{p_0, p_1, v\}$; by the definition of an induced path, $(p_0, v) \notin E$. However, $(p_0, v) \in E$ as $v \in U(C)$, so we have reached a contradiction. Therefore, as there is no induced P_3 having v as an endpoint, we may use rule **RA** at any point during to remove v. Thus we can use a rule in the case $v \in U(C)$.

Case 2: $v \notin U(C)$. We claim that there is no induced P_3 containing v as a midpoint. Suppose to the contrary that v is the midpoint of some induced $P_3 = \{p_0, v, p_2\}$; by the definition of an induced path, $(p_0, p_2) \notin E$. By construction, the only non-edges of any component are between the vertices of C_i and C_j for $i \neq j$. However, the only vertices adjacent to vertices of both C_i and C_j for some i, j are those in $U(C)$. Therefore, we must have that $v \in U(C)$, which is a contradiction. Therefore, there as there is no induced P_3 with v as a midpoint, we may use rule **RB** to remove v at any point. Thus, we can use a rule in the case $v \notin U(C)$.

Therefore, if a graph was constructed using operations **O1** and **O2**, it has an elimination ordering using rules **RA** and **RB** such that at every stage of the ordering, every remaining vertex is eligible to be removed.

Conversely, suppose that G has an elimination ordering using rules **RA** and **RB** such

that at every stage of the ordering, every remaining vertex is eligible to be removed. In particular, this means that before the first vertex is eliminated, any vertex in the graph can be eliminated. We will show how we can reconstruct each component using operation **O1**, which implies that G can be constructed using operation **O2** over all components. We claim that we can partition the vertices of each component based on which rule can be used to remove each vertex of the component. Fix a component C of G.

Claim 2.3.3. *If vertex $v \in C$ can be removed using rule* **RA**, *then $v \in U(C)$.*

Proof of claim: Suppose that a vertex $v \in C$ can be removed using rule **RA**. Since rule **RA** can be applied to v, v is not the endpoint of any induced P_3. We claim that $v \in U(C)$. Suppose to the contrary that $u \notin U(C)$; then there is at least one vertex $u \in C$ such that $u \notin N(v)$. Let P be a shortest (u,v)-path, which must exist since u and v are in the same component. By the assumption that rule **RA** is applicable to remove v, $|P| \neq 3$. Therefore, $|P| \geq 4$. However, taking $P = \{v, p_1, p_2, \ldots, p_i = u\}$ we see that v is the endpoint of the induced $P_3 = \{v, p_1, p_2\}$ (where it must be that $(p_2, v) \notin E$ as otherwise P was not a shortest path). This contradicts the applicability of rule **RA**. Therefore, $v \in U(C)$.

Therefore, all vertices which can be removed by rule **RA** are in $U(C)$, and necessarily $U(C)$ is a clique. If C is a clique, we are done with this component, as all vertices can be removed by rule **RA**. We may therefore assume that C is not a clique.

Suppose that a vertex $v \in C$ can only be removed using rule **RB**. Since rule **RB** can be applied to v, v is not the midpoint of any induced P_3.

We claim that if $v \in U(C)$, then C is a clique. Suppose to the contrary that $(x,y) \notin E$ for some vertices $x, y \in C$. Since $v \in U(C)$, $(v,x) \in E$ and $(v,y) \in E$. However, now $\{x, v, y\}$ induce a P_3, contradicting the fact that rule **RB** is applicable to v. Therefore, C was a clique, contradicting our assumption, and it must be that $v \notin U(C)$.

We establish the following useful claim.

Claim 2.3.4. *Let $v \in C \cap U(C)$ be such that v can be removed by rule* **RB**. *If $u \in N(v) \cap U(C)$, then u is not adjacent to any vertex $w \notin N(v)$.*

Proof of claim: Suppose to the contrary that $(u,w) \in E$ but $(w,v) \notin E$. Then $\{w, u, v\}$ induce a P_3, and therefore u cannot be removed by rule **RB**. By assumption, every vertex can be removed at any stage; if u cannot be removed by rule **RB**, it must be possible to remove it using rule **RA**. However, by Claim 2.3.3, this means that $u \in U(C)$, a contradiction to our choice of u. Therefore, no such vertex w exists where $(u,w) \in E$ but $(w,v) \notin E$.

Using the previous claim, we now show that $N(v) \cap U(C)$ induces a complete graph.

- Suppose that $N(v) \cap U(C) = \emptyset$. Then v is a K_1, and by definition of $U(C)$, v is adjacent to every vertex of $U(C)$.
- Suppose that $N(v) \cap U(C) = \{u\}$ for some vertex u. Then $(u,v) \in E$, and by definition of $U(C)$, both u and v are adjacent to every vertex of $U(C)$. By Claim 2.3.4, u is not adjacent to any vertex $w \notin N(v)$. Therefore, it must be that $\{u,v\}$ is a K_2 with no neighbours except for $U(C)$.
- Finally, suppose that $|N(v) \cap U(C)| \geq 2$, and consider any two distinct vertices x,y in $N(v) \cap U(C)$. By definition of $N(v)$, $(x,v) \in E$ and $(y,v) \in E$. By Claim 2.3.4, neither x nor y is adjacent to any vertex $w \notin N(v)$. We claim that $(x,y) \in E$. Suppose to the contrary that this is not the case, i.e., $(x,y) \notin E$. Then since $x \in N(v)$ and $y \in N(v)$, $\{x,v,y\}$ induce a P_3 with v as the center, contradicting the fact that rule **RB** applied to v. Therefore, we must have that $(x,y) \in E$. Since x and y were arbitrary, we must have that $N[v] \cap U(C)$ is a complete graph on $|N[v] \cap U(C)|$ vertices where each vertex is additionally adjacent to every vertex in $U(C)$.

Let k be the number of distinct sets of $N[v] \cap U(C)$ over all vertices $v \in C$ that can be removed by rule **RB**. For each v which can be removed by rule put v into a set C_i $1 \leq i \leq k$ based on $N[v] \cap U(C)$ (so that all vertices v in C_i have the same neighbourhood $N[v] \cap U(C)$). Therefore, we can partition C into vertices which can be removed by rule **RA** which are adjacent to all vertices in C, as well disjoint complete graphs $C_1,\ldots,C_k$ for some $k \geq 0$. Since C was arbitrary, every component can be constructed using operation **O1**. Therefore, the graph can be constructed from the components using operation **O2**, as required. □

2.3.2 Asteroidal-Triple-Free Graphs

We now define AT-free graphs. An *asteroidal triple* (AT) is a triple of independent vertices x,y,z such that between every pair of vertices in the triple, there is a path that misses (i.e., does not intersect the closed neighbourhood of) the third. An example of an asteroidal triple is shown in Figure 2.8. A graph is *asteroidal-triple-free* (*AT-free*) if it does not contain any asteroidal triples.

AT-free graphs generalize interval graphs but are incomparable to chordal graphs. To see that AT-free graphs are incomparable to chordal graphs, note that a chordless cycle on five vertices is AT-free but not chordal, while a claw ($K_{1,3}$) where each edge has been subdivided is chordal but not AT-free. The fact that the chordless cycle on five vertices

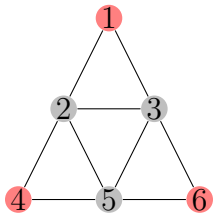


Figure 2.8: A graph with an asteroidal triple $(1; 4; 6)$.

is AT-free also implies that the class of AT-free graphs is not contained in the class of so-called *perfect* graphs (see, e.g., Golumbic [65] for more on perfect graphs).

We now define some terms which are related to AT-free graphs.

> Two vertices $y; z$ of G are said to be *unrelated with respect to a vertex* x if there exists an $(y; x)$-path P that misses z (i.e., $P \setminus N[z] = ;$) and a $(z; x)$-path Q that misses y (i.e., $Q \setminus N[y] = ;$).
>
> A vertex x is *admissible* if no pair of vertices is unrelated with respect to x.

2.3.3 Bipartite Permutation Graphs

A graph is a *permutation* graph if each vertex can be represented by a point on each of two parallel lines and a straight line segment joining the points, and two vertices are adjacent if and only if their corresponding line segments intersect. A graph is a *bipartite permutation* graph if it is both a bipartite graph and a permutation graph.

A *proper interval bigraph* is a bipartite graph that is AT-free and contains no induced cycle of size greater than four (see, e.g., Brandstädt et al. [18]). For this reason, proper interval bigraphs are also known as AT-free bigraphs. Equivalently, a proper interval bigraphs is a bigraph in which each vertex v may be identified with a interval I_v, coloured with one of two colours, of the real line, such that $(u; v)$ 2 E if and only if $I_v \setminus I_v$ 6= $;$ and the colour of I_u and I_v differs. Proper interval bipartite graphs are precisely the bipartite permutation graphs (see, e.g., Hell and Huang [80]).

Let $G = (A; B; E)$ be a bipartite graph, and let ^{A} be an ordering of the partite set A and ^{B} be an ordering of the partite set B. Some orderings of bipartite graphs have special properties:

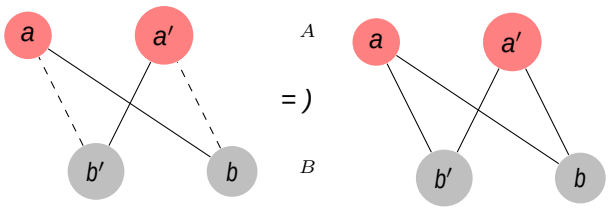


Figure 2.9: The property required of a strong ordering for a bipartite permutation graph. The top vertices in each image (red vertices) are in one partite set A, while the bottom vertices in each image (gray vertices) are in the other partite set B of a bipartite graph $G = (A; B; E)$. A strong ordering requires that if the cross edges are found for orderings ^{A} and ^{B}, then the parallel edges shown in the right image must also exist.

A *strong ordering* consists of an ordering ^{A} of A and an ordering ^{B} of B such that for all $(a; b); (a^0; b^0)$ 2 E, where $a; a^0$ 2 A and $b; b^0$ 2 B, if $a <_A a^0$ and $b^0 <_B b$, then $(a; b^0)$ 2 E and $(a^0; b)$ 2 E (see Figure 2.9). We will use $(^A; ^B)$ to denote a strong ordering of a bipartite graph.

An ordering ^{A} of A has the *adjacency property* if, for every b 2 B, $N(b)$ consists of vertices that are consecutive in ^{A}.

The ordering ^{A} has the *enclosure property* if, for every pair $b; b^0$ of vertices of B with $N(b)$ $N(b^0)$, the vertices of $N(b^0) \cap N(b)$ appear consecutively in ^{A}, implying that b is adjacent to the leftmost or rightmost neighbour of b^0 in ^{A}.

Spinrad et al. [129] showed that bipartite permutation graphs are exactly the bipartite graphs which have a strong ordering.

Theorem 2.3.5 (Theorem 1, Spinrad et al. [129]). *The following statements are equivalent for any bipartite graph $G = (A; B; E)$.*

1. *G is a bipartite permutation graph.*
2. *G has a strong ordering.*
3. *There exists an ordering of A which has the adjacency property and the enclosure property.*

As observed by at least Chang et al. [22] and Heggernes et al. [79], the next lemma follows from the proof of Theorem 2.3.5 when G is connected. The subsequent theorem shows that a strong ordering where the ordering of each part of the graph has both the adjacency and enclosure properties can be found in linear time.

Lemma 2.3.6 (Lemma 2.2, Heggernes et al. [79]). *Let $(\ ^A;\ ^B)$ be a strong ordering of a connected bipartite permutation graph $G = (A; B; E)$. Then both ^{A} and ^{B} have the adjacency property and the enclosure property.*

Theorem 2.3.7 (Theorem 3, Chang et al. [22]). *A strong ordering (such that the orderings of each part of the graph have the enclosure and the adjacency property) of a connected bipartite permutation graph can be generated in linear time.*

2.4 Complexity Classes

In this section we discuss some complexity classes which will appear in the thesis. We will frequently use the classes P and NP, which are the usual classes of problems solvable in deterministic polynomial time and problems with "yes"-instances verifiable in nondeterministic polynomial time, respectively.

In addition to P and NP, we will also use the related class coNP=poly, which we define[2] below, along with the related class coNP. Informally, the class coNP=poly is the class of problems which can be solved in coNP by a Turing machine which also has a (polynomially-bounded) "advice" string. The Turing machine can use the "advice" string to reach a conclusion for the instance it is solving. It is important to note that the "advice" string depends only on the size of the input. For an introduction to Turing machines and complexity classes, see, e.g., Sipser [128].

Definition 2.4.1 (Complexity class coNP). *We say that a language L belongs to the complexity class* coNP *if there is a Turing machine M such that*

- *Machine M, when given input x of length n has to decide whether $x \in L$. Machine M works in* co-nondeterministic polynomial time*. That is, it makes a polynomial number of steps that may be chosen co-nondeterministically: If $x \in L$, the algorithm should derive this conclusion for* every *possible run, whereas if $x \notin L$, then* at least one *run needs to finish with this conclusion.*

[2] We present the classes in the style of Cygan et al. [41].

In short, a problem is in the class coNP if "no"-instances have a polynomialy-sized proof that the instance is a "no"-instance (such proof must also be verified in polynomial time). The next definition adds the "advice" string to the class coNP.

Definition 2.4.2 (Complexity class coNP=poly). *We say that a language L belongs to the complexity class* coNP=poly *if there is a Turing machine M and a sequence of strings $(\ _n)_{n=0;1;2;:::}$, called the* advice*, such that*

- *Machine M, when given input x of length n, has access to string $\ _n$ and has to decide whether $x \ 2 \ L$. Machine M works in* co-nondeterministic polynomial time.
- $j \ _n j \quad p(n)$ *for some polynomial $p(\ )$.*

We now make some remarks about these classes. First, it is unknown whether P = NP and it is widely believed that P 6= NP (see, e.g., Fortnow [54] for a brief overview of the question). Second, observe that the advice for coNP=poly depends only on the size of the input x, and in particular that for two inputs x and x^0 of the same size, the machine M must use the same advice string. It is also unknown if coNP = NP, and the assumption that NP 6 coNP=poly is a stronger statement than coNP 6= NP due to the availability of the advice string. It is known that if NP coNP, then the polynomial hierarchy collapses to its third level (which does not imply that P = NP) (see, e.g., Cai et al. [20]). We will use the assumption NP 6 coNP=poly in one new theorem (Theorem 4.7.5).

2.4.1 Parameterized Complexity Classes

An instance of a parameterized problem is a pair $(I; k)$ where I is the input to the problem and $k \ 2$ N is the parameter. A parameterized problem is *xed-parameter tractable* (FPT) if there is an algorithm that solves instances $(I; k)$ in time $f(k)jIj^c$ where c is a constant, jIj indicates the size of the input, and $f(k)$ is a computable function of k. The class FPT denotes all fixed-parameter tractable problems. A parameterized problem L is *FPT reducible* (through an *FPT reduction*) to a parameterized problem L^0 if there is a mapping R from instances of L to instances of L^0 and (i) $(I; k) \ 2 \ L$ if and only if $(I^0; k^0) = R(I; k) \ 2 \ L^0$; (ii) $k^0 \quad g(k)$ for a computable function $g(k)$; and (iii) R can be computed in time $O(f(k)jIj^c)$ for a computable function $f(k)$ and a constant c.

A parameterized problem can always be reduced to a *kernel*. A kernelization algorithm takes an instance $(I; k)$ of a parameterized problem L and transforms it in polynomial time $O(jIj^c)$ into an instance, called the kernel, $(n^0; k^0)$ such that $(I; k) \ 2 \ L$ if and only

if $(I^0; k^0)$ 2 L, but $jI^0j; k^0$ $g(k)$ for some computable function $g(k)$. The function $g(k)$ is the size of the kernel. Downey and Fellows [47] showed that a decidable parameterized problem has a kernel if and only if it is in FPT, but generally only polynomial-sized kernels are of interest (see, e.g., Fellows et al. [50]).

An alternative notion of parameterized complexity is captured by the class XP, which denotes the class of parameterized problems that can be decided in time $O(n^{g(k)})$. However it is known that FPT (XP (e.g., Downey and Fellows, Ch. 15 [45]), so showing that a parameterized problem is in FPT is stronger than showing that it is in XP.

Not all pairs of problems and parameters are equal. For a given problem, the choice of parameter may affect the problem's complexity. The use of some parameters may lead to FPT algorithms, while the use of other parameters may result in instances which are provably hard, which we do not discuss here (see, e.g., Downey and Fellows [45] for more on parameterized hardness).

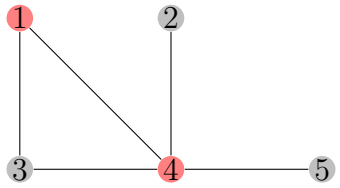
2.5 Graph Parameters

As discussed in Section 2.4.1, parameterized problems take an input $(I; k)$ where I is the instance of the problem and k is the parameter. The choice of parameter can vary widely and affect the existence of an FPT algorithm for the given pair $(I; k)$. Often, the parameter is natural, like the size of a solution desired for the problem on the supplied instance. For problems on graphs, another choice is often some parameter which somehow describes the structure of the graph in the supplied input I.

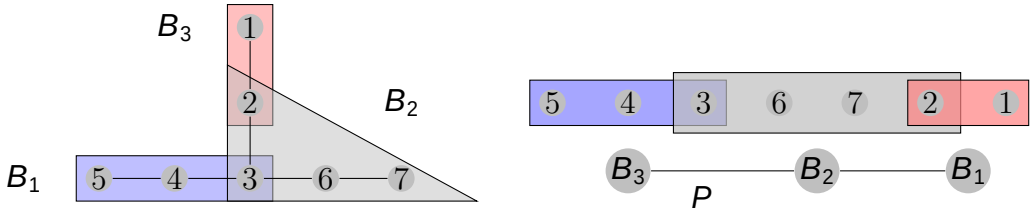
We define the following parameters that we will reference in this work. The parameters are illustrated in Figures 2.10 and 2.11.

A *vertex cover* is a set C $V(G)$ such that for every $(u; v)$ 2 $E(G)$, either u 2 C or v 2 C (or both). The set C is said to *cover* the edges of the graph and is sometimes referred to as a "cover." The *vertex cover number* of G, denoted $vc(G)$, is defined to be the size of a smallest vertex cover of G. See Figure 2.10a for an illustration.

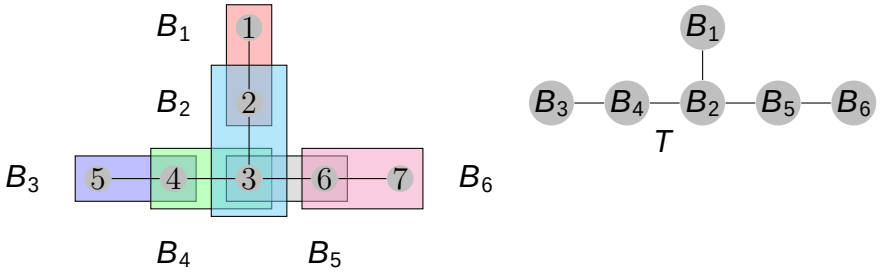
A graph has *pathwidth* at most k, denoted $pw(G)$ if the vertices can be put into bags with size at most $k + 1$ (so that every vertex is in at least one bag), every edge has both endpoints in some bag, and the bags form a path P where, if B_i and B_j both contain a vertex v, all bags on the (unique) path from B_i to B_j in P also contain v. Thus the bags can be linearly ordered so that the bags containing each vertex



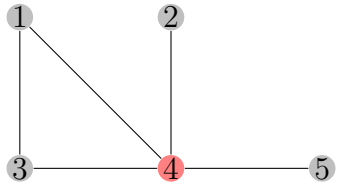
(a) An illustration of a vertex cover of a graph G. The set $\{1, 4\}$ is a vertex cover of G as every edge has at least one endpoint as either 1 or 4. Note that $vc(G) > 1$ since no vertex is incident with every edge. In this case, $vc(G) = 2$. This minimum vertex cover is not unique.



(b) An illustration of a path decomposition of a graph G, with three bags B_i indicated (left). Each bag has at most 4 vertices, every edge in the graph has both endpoints in at least one bag, and the bags containing each vertex form a sub-path of P (right). Therefore, for this example, $pw(G)$ 3.



(c) An illustration of a tree decomposition of a graph G, with six bags B_i indicated (left). Each bag has at most 2 vertices, every edge in the graph has both endpoints in at least one bag, and the bags containing each vertex form a subtree of T (right). Therefore, for this example, $tw(G)$ 1.



(d) An illustration of a twin cover of a graph G. The set $\{4\}$ is a minimum twin cover of G as every edge has at least one endpoint as 4, except the edge $(1, 3)$, which is between two true twins ($\{1, 3\}$). The minimum twin cover is unique in this example (this can be veri ed by checking every other vertex). In this case, $tc(G) = 1$.

Figure 2.10: Illustrations of some graph parameters discussed in this work.

are consecutive. The bags constitute a *path decomposition* of the graph. See Figure 2.10b for an illustration.

A graph has *treewidth* at most k, denoted $tw(G)$ if the vertices can be put into bags B_i with size at most $k+1$ (so that every vertex is in at least one bag), every edge has both endpoints in some bag, and the bags form a tree T where, if B_i and B_j both contain a vertex v, all bags on the (unique) path from B_i to B_j in T also contain v. The bags constitute a *tree decomposition* of the graph. See Figure 2.10c for an illustration. See, e.g., Bodlaender [10] or Downey and Fellows [45] for more on treewidth.

A *twin cover* of G is a set T V such that for every edge $(u;v)$ 2 E, either $fu;vg \setminus T$ $6=$ $;$ or u and v are true twins (Ganian [57]). The minimum size of a twin cover is denoted $tc(G)$. A twin cover is similar to a vertex cover, except that edges between true twins do not need to have an endpoint in the set T. Every vertex cover is a twin cover, and therefore $tc(G)$ $vc(G)$. However, the difference $vc(G)$ $tc(G)$ may be arbitrarily large (Ganian [57]). See Figure 2.10d for an illustration.

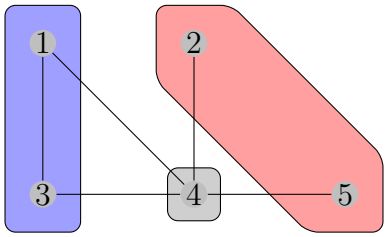
A graph $G = (V;E)$ has *neighbourhood diversity* at most k, if V can be partitioned into at most k sets, such that all the vertices in each part is a set of twins (Lampis [94])[3]. The smallest number of sets required to partition G is the neighbourhood diversity of the graph and is denoted $nd(G)$. Each set is therefore either an independent set (if it is a set of false twins) or a clique (if it is a set of true twins). See Figure 2.11a for an illustration.

We now define the modular width of a graph. A *module* is a set M $V(G)$ such that for any x 2 $V(G)$ n M, x is either adjacent to all vertices of M or none of them; a module is *trivial* if $jMj = 1$. A graph G has *modular width* at most k if: G has at most one vertex; G is a disjoint union of graphs of modular width at most k; G is the complete join of graphs of modular width at most k; or the vertex set of G can be partitioned into p k modules $M_1;:::;M_p$ such that $G[M_i]$ is of modular width at most k for 1 i p. The modular width of a graph is denoted $mw(G)$. An example is shown in Figure 2.11b. For more on modular width, see, e.g., Fomin et al. [53] or Gajarský [56].

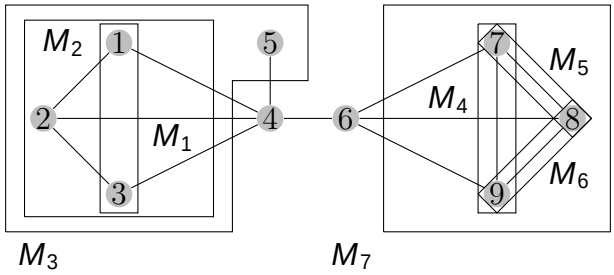
The *clique width* of a graph G, denoted[4] $cwd(G)$, is the minimum number of labels needed to construct G using the following four operations: (C1) create a new vertex

[3] Following Ganian [58], we do not consider coloured graphs, as the original de nition does.

[4] We use $cwd(G)$ instead of $cw(G)$ to avoid confusion with the cutwidth of a graph.



(a) An illustration of neighbourhood diversity. The enclosed areas partition the vertices of G into sets of true twins $\{1, 3\}$ and false twins $\{2, 5\}$ and $\{4\}$. In this case, $nd(G) = 3$.



(b) A graph G with non-trivial modules M_1, M_2, M_3, M_4, M_5, M_6, and M_7. Since G can be partitioned into modules M_3, $\{4\}$, $\{6\}$, M_7 and each of the graphs $G[M_3]$ and $G[M_4]$ have modular width at most 4, $mw(G) = 4$.

Figure 2.11: Illustrations of more graph parameters discussed in this work.

v with label i; (C2) the disjoint union of two labelled graphs; (C3) join by an edge every vertex labelled i to every vertex labelled j, where $i \neq j$; and (C4) rename label i to label j. An example is shown in Figure 2.12. For more on clique width, see, e.g., Corneil and Rotics [37].

Some of the above parameters are very well studied, like the vertex cover number, pathwidth, or treewidth of a graph. Others, like the twin cover number or neighbourhood diversity, are less studied. We include twin cover number and neighbourhood diversity because we will describe special cases of them (in Sections 2.5.1 and 2.5.2). The remaining two parameters, modular width and clique width, are presented in order to establish the relationship between other parameters, which we now formally describe.

Graph parameters are often compared asymptotically. We say that a parameter x is at least as general as parameter y if every graph class where y is bounded also has bounded x value. For example, it is well-known that if a graph has a bounded vertex cover number, then it also has a bounded pathwidth (see, e.g., Sasák [125]). Figure 2.13 shows the relation

1. Let A be the disjoint union of the following graphs:
   (a) Create a new vertex with label 1 (vertex d)
   (b) Create a new vertex with label 3 (vertex c)
2. Join all vertices in A with label 1 to those vertices in A with label 3 to get A'.
3. Let B be the disjoint union of the following graphs:
   (a) Create a new vertex with label 1 (vertex a)
   (b) Create a new vertex with label 2 (vertex b)
4. Join all vertices in B with label 1 to those vertices in B with label 2 to get B'.
5. Let C be the disjoint union of A' and B'.
6. Join all vertices in C with label 2 to those vertices in A with label 3 to get $G = P_4$.

Figure 2.12: A graph $G = P_4$ constructed according to the operations for clique width. Since we only used three labels, $cwd(G)$ 3. In this example, the graph was constructed without renaming a label.

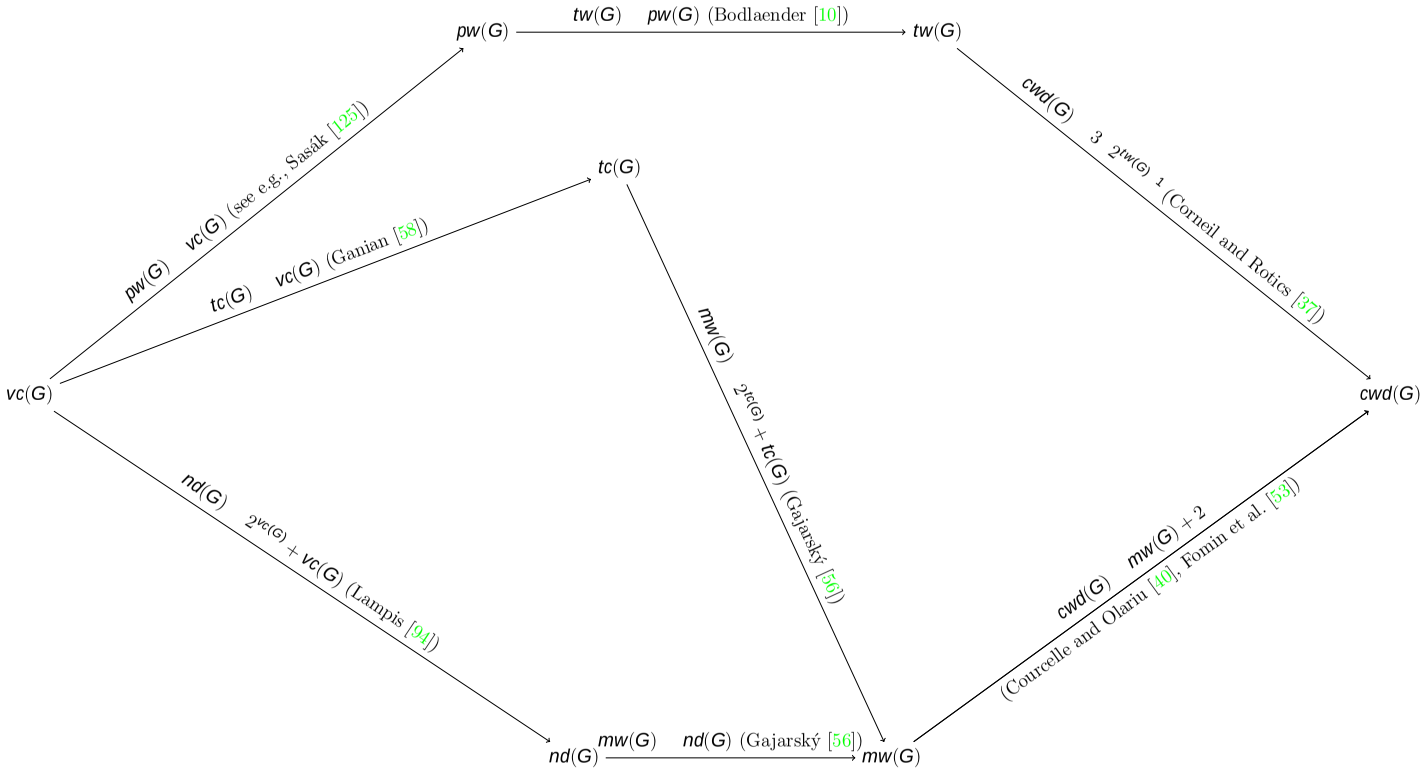


Figure 2.13: The relationship between several parameters for graph problems. An arrow $f(G) \; ! \; g(G)$ means that the parameter $f(G)$ is at least as general as the parameter $g(G)$.

between the parameters listed above; the figure is drawn this way as we will update it with more parameters throughout the thesis. The layout of the parameters in Figure 2.13 is significant: a parameter x to the left of another parameter y is at least as general as y (if there is a left-to-right path between the parameters, denoted by the arrows). From the picture, it is clear that all parameters in this work are at least as general as clique width. The parameters twin cover number and neighbourhood diversity are also both at least as general as the modular width of a graph. If a parameter x is at least as general as a parameter y, and there is an FPT algorithm for a problem using the parameter y, then there is also an FPT algorithm for the parameter x.

The following lemma shows a relationship between the twin cover number of a graph and its vertex cover number, and will be helpful.

Lemma 2.5.1 (Lemma 2, Ganian [58])**.** *Given a graph G and a graph G' obtained by deleting all edges $(u;v)$ such that u and v are true twins, the vertex cover number of G' equals the twin cover number of G.*

Lemma 2.5.2. *Let G be a graph. If T is a twin cover of G, then the components of $G \quad T$ are sets of true twins.*

Proof. Let C be a component of $G \quad T$. The result holds trivially if $jCj = 1$; suppose instead that $jCj > 1$. Since C is a component, it is connected, i.e., there is an edge with both endpoints $(u;v)$ in C (and neither in T). By definition of a twin cover, the only edges which do not have an endpoint in T are between true twins. Therefore, $fu;vg$ are true twins; since the edge in C was arbitrary, the component consists of true twins. □

2.5.1 Edge Clique Cover

The *edge clique cover number* of G, denoted[5] $cc(G)$, is defined to be the minimum number of cliques required to cover all edges of G. Here, a clique C *covers* an edge $(u;v)$ if $u;v \ 2 \ C$. See Figure 2.14 for an illustration.

In this section, we provide some results for the edge clique cover number which are not clear from the existing literature. We start by showing that the edge clique cover number is incomparable to the vertex cover number as a parameter.

Proposition 2.5.3. *There exist graph classes with bounded edge clique cover number but unbounded vertex cover number.*

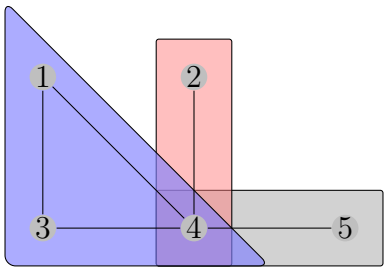


Figure 2.14: An illustration of an edge clique cover of a graph G. Each shaded region is a clique, and together they cover the graph as every edge in the graph is entirely contained within one clique. In this case, $cc(G) = 3$.

Proof. Consider the class of complete graphs. Each complete graph has one maximal clique, so $cc(G) = 1$ for each graph G in the class. However, the vertex cover number of the class of graphs is unbounded, since the cliques can be arbitrarily large. □

We show that a graph with a bounded vertex cover number does not necessarily have a bounded edge clique cover number, but if a graph has bounded edge clique cover number it has bounded neighbourhood diversity.

Proposition 2.5.4. *There exist graph classes with bounded vertex cover number but unbounded edge clique cover number.*

Proof. Let G be a star with ℓ pendants (vertices of degree 1). We will show that the class of stars for each integer $\ell \geq 3$ has bounded vertex cover number but unbounded edge clique cover number.

Let G be a star with $\ell \geq 3$ pendants. Taking $C = \{v\}$ where $v \in V(G)$ is such that $\Delta(G) = d_G(v)$ is a vertex cover of size 1, which is minimum, since if any other vertex is taken, we also need v or at least another pendant vertex. However, every one of the ℓ edges requires its own clique to be covered, so $cc(G) \geq \ell$. Since ℓ is not a function of $vc(G)$, the proposition is proved. □

Proposition 2.5.5. *For any graph G, $nd(G) \leq 2^{cc(G)}$; i.e., if $cc(G)$ is bounded, then $nd(G)$ is bounded.*

[5]Other authors, e.g., Fellows et al. [49], denote this parameter using $ecc(G)$. We change it to avoid confusion with the eccentricity of a graph.

Proof. Any vertex has a neighbourhood contained in a subset of the cliques used to cover the graph. Therefore if the graph can be covered by k cliques, there are at most 2^k distinct neighbourhoods. To see this, observe that every vertex v is in a subset of the cliques covering the graph. Moreover, the neighbourhood for each vertex v can be partitioned into a subset of the cliques covering the graph. The result follows from the fact that there are at most 2^k subset of the covering cliques. □

Figure 2.15 shows the relationship between $cc(G)$ and the other parameters discussed in this work.

Theorem 2.5.6 (Corollary 2 and discussion in section 3 in Gramm et al. [71])**.** *There is an FPT algorithm to determine if $cc(G)$ k for any graph G where the parameter is k. Moreover, there is also an FPT algorithm with parameter k for generating the clique cover.*

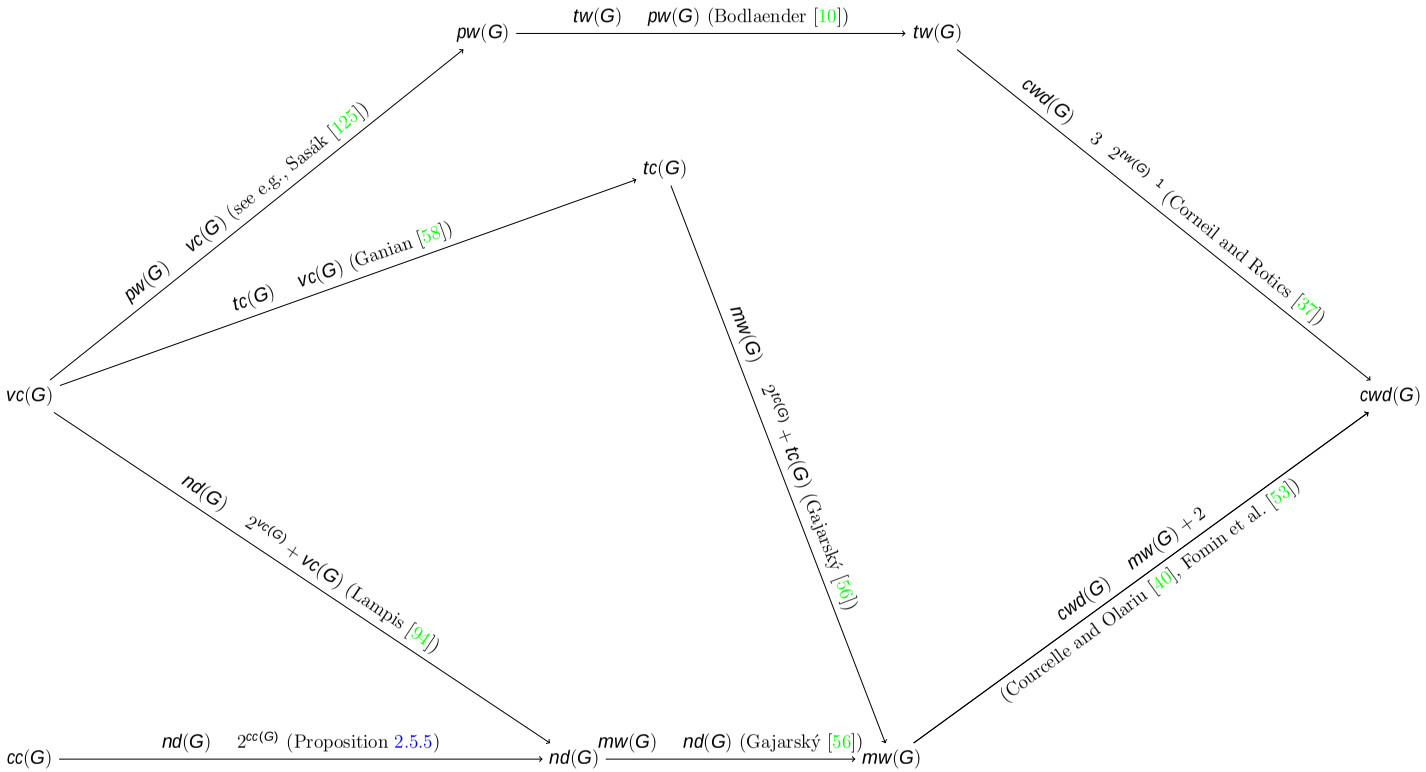


Figure 2.15: The relationship between edge clique cover number and other parameters for graph problems.

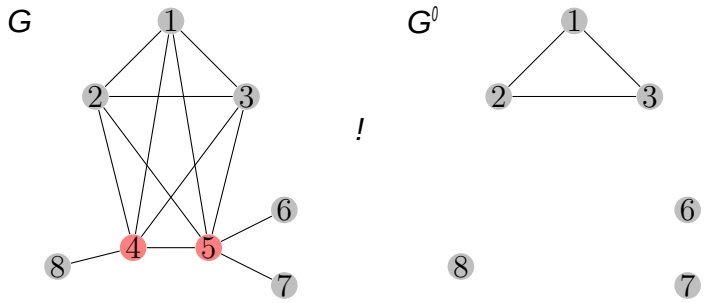


Figure 2.16: An illustration of the restricted twin cover parameter. The graph G, left, has a twin cover $T = \{4, 5\}$ of size 2. After deleting T, there is one non-trivial component in G' (right). Therefore in this example, $rtc(G) \quad |T| + 1 = 3$. By trying all possible twin covers of G, one can verify that $rtc(G) \quad 3$.

2.5.2 Restricted Twin Cover

In this subsection, we define a new special case of the neighbourhood diversity parameter, called the restricted twin cover number.

Let $G = (V, E)$ be a graph, and let T be the set of twin covers of G. We define the *restricted twin cover number* of G, denoted $rtc(G)$, as the smallest integer k such that there is a twin cover $T \ 2 \ T$ with $k = |T| + q$ where q is the number of non-trivial components of $G \quad T$. An example is shown in Figure 2.16.

The restricted twin cover is related to other parameters. We establish this relationship by considering several parameters discussed in the previous sections.

By definition, we have that $tc(G) \quad rtc(G)$ for any graph G. The opposite direction is not true, as we show next.

Proposition 2.5.7. *There exist graph classes with bounded twin cover number but unbounded restricted twin cover number.*

Proof. Consider a graph obtained by taking ℓ disjoint copies of K_2 and adding all edges from these 2ℓ vertices to a new vertex u. Note that each such graph has exactly one universal vertex. We will show that the class of graphs obtained by this construction for each integer $\ell \quad 3$ has bounded twin cover number but unbounded restricted twin cover number.

Observe that taking the set $T = \{u\}$ is a twin cover of size 1, since all edges between u and the vertices of a K_2 are covered by u, while edges within a K_2 are between true twins. Thus any graph G in the class, $tc(G) \quad 1$.

We now show that for any graph in the class, $rtc(G)$ may be arbitrarily large. Let G be a graph in the class, and let T be a twin cover of G such that $rtc(G) = |T| + q$ for some $q \quad 0$ such that there are q non-trivial components of $G \quad T$.

Claim 2.5.8. *We may assume that $u \in T$ for $u \in U(G)$.*

Proof of claim: Assume to the contrary that this is not the case, i.e., that $u \notin T$.

For every vertex $v \in G \setminus U(G)$, u and v are not true twins: there are at least 3 copies of K_2 which are all adjacent to u, and so there is a vertex v' in a K_2 which does not contain v such that $(v', v) \notin E(G)$ (but $(u, v') \in E(G)$). Since T is a twin cover of G, for every vertex $v \in G \setminus U(G)$, (u, v) must have at least one endpoint in T.

By the previous observation, the edge (u, v) is not between true twins. Therefore, each such edge must have $v \in T$ as $u \notin T$. Moreover, $G \quad T$ has only one vertex, u, and therefore contains no non-trivial components. Thus we have that $q = 0$.

Since each edge of the 2ℓ edges (u, v) must be covered by T and $u \notin T$, $|T| \quad 2\ell$. Therefore $rtc(G) = |T| + 0 \quad 2\ell$.

However, by taking the set $T' = \{u\}$, we obtain a twin cover of G such that $|T'| = 1$. Note that $G \quad T'$ has ℓ non-trivial components (the ℓ copies of K_2). Thus $rtc(G)$ $|T'| + \ell = 1 + \ell < 2\ell \quad rtc(G)$ as $\ell \quad 3$, a contradiction as T was chosen so that $rtc(G)$ is minimized. Thus the claim is proved.

Claim 2.5.9. *We may assume that $|T| = 1$.*

Proof of claim: Suppose that $|T| \quad 1$. The set T must contain u and $|T| \quad 1$ vertices of $G \setminus U(G)$ by the Claim 2.5.8. That is, $|T|$ contains $|T| \quad 1$ vertices contained in the copies of K_2.

First, we show that no two vertices v, v' of $T \setminus \{u\}$ are adjacent. Suppose to the contrary that such a pair exists, and consider $T' = T \setminus \{v\}$. Then $|T'| = |T| \quad 1$ and T' is a twin cover: the edge (v, v') is covered as its endpoints are true twins, and the edges (u, v) and (u, v') are covered as $u \in T'$. Now $G \quad T'$ has q non-trivial components: the vertex v is present, but is a trivial component. Thus $rtc(G) = |T| + q > |T| \quad 1 + q = |T'| + q$, which is a contradiction as $rtc(G)$ is the minimum over all twin covers of G. Thus no two vertices v, v' of $T \setminus \{u\}$ are adjacent.

Let $v \in T \setminus \{u\}$, and consider $T'' = T \setminus \{v\}$. Then, observe that $G \setminus T''$ has one more non-trivial component than $G \setminus T$, as the K_2 containing v now remains after removing T''. Thus we have

$$\begin{aligned} rtc(G) &= |T| + q \text{ for some } q \geq 0 \\ &= |T| + q \quad 1 + 1 = |T| \quad 1 + q + 1 \\ &= |T''| + (q+1). \end{aligned}$$

Therefore, we could use T'' instead of T to achieve the restricted twin cover number of G.

Since we may assume that $|T| = 1$, $G \setminus T$ has ℓ non-trivial components (as $T = \{u\}$ by Claims 2.5.8 and 2.5.9), which may be arbitrarily large. □

A bound on $rtc(G)$ implies a bound on the neighbourhood diversity of a graph, as is shown in the next proposition.

Proposition 2.5.10. *For any graph G, $nd(G) \leq (1 + rtc(G))2^{rtc(G)} + rtc(G)$; i.e., if $rtc(G)$ is bounded then $nd(G)$ is bounded.*

Proof. Suppose that G is a graph that has $rtc(G) = k$ for some $k \geq 0$. Let T be a twin cover such that $rtc(G) = |T| + q$ for some $q \geq 0$ where there are q non-trivial components of $G \setminus T$.

Every component of $G \setminus T$ has an open neighbourhood which is a subset of T. Moreover, every vertex in a non-trivial component of $G \setminus T$ has the same neighbourhood and each non-trivial component is a clique (Lemma 2.5.2). By definition, there are q non-trivial components of $G \setminus T$. The closed neighbourhood of each non-trivial component consists of itself and a subset of the vertices of T. There are at most $2^{|T|}$ possible subsets of T. Thus we can partition the vertices in the non-trivial components of $G \setminus T$ into at most $q \cdot 2^{|T|}$ different cliques.

Similarly, each vertex in a trivial component of $G \setminus T$ also has a neighbourhood which is a subset of T. Thus we can partition the vertices in the trivial components of $G \setminus T$ into at most $2^{|T|}$ distinct independent sets, based on the neighbourhoods of these components.

Lastly, we can partition the $|T|$ vertices of T into $|T|$ independent sets, each of size one.

Therefore we can partition G into at most $2^{|T|} + q \cdot 2^{|T|} + |T| \leq 2^{rtc(G)} + rtc(G) \cdot 2^{rtc(G)} + rtc(G) \leq (1 + rtc(G))2^{rtc(G)} + rtc(G)$ different sets, each of which is either an independent set or a clique. □

On the other hand, a bound on $nd(G)$ does not imply a bound on $rtc(G)$, as shown next.

Proposition 2.5.11. *There exist graph classes with bounded neighbourhood diversity but unbounded restricted twin cover number.*

Proof. Consider the class of graphs obtained by adding all edges between an independent set I_a and a complete graph K_b for $a \geq 2$ and $b \geq 2$.

Let G be a graph in this class, In this case, we can partition I_a into one set and K_b into another to satisfy the definition of neighbourhood diversity, so $nd(G) \leq 2$.

On the other hand, every twin cover of such a graph requires $\min\{a, b\}$ vertices, i.e., $tc(G) = \min\{a, b\}$.

First, we establish that $tc(G) \leq \min\{a, b\}$. To see this, note that if $a \leq b$, then by taking K_b as a twin cover, every edge has an endpoint in K_b, so we can use the b vertices of K_b. If instead $b \leq a$, then I_a is a twin cover: every edge missing are those within K_b, which are true twins, and we can use the a vertices of I_a.

Next, we establish that $tc(G) \geq \min\{a, b\}$ in the following claim.

Claim 2.5.12. $tc(G) \geq \min\{a, b\}$

Proof of claim: We prove the claim by contradiction. Suppose to the contrary that there is a twin cover T of G such that $|T| < \min\{a, b\}$.

Since $|T| < \min\{a, b\}$, $T < \min\{a, b\} \leq a$, and there is at least one vertex $v \in I_a$ which is not in the set T (since T can contain at most $a - 1$ vertices of I_a). Similarly, $T < \min\{a, b\} \leq b$, and there is at least one vertex $u \in K_b$ which is not in the set T (since T can contain at most $b - 1$ vertices of I_b). Since $a \geq 2$, there is a vertex $v' \in I_a \setminus \{v\}$; since I_a is an independent set, $(v, v') \notin E(G)$. By construction every vertex in I_a is adjacent to every vertex of K_b, so $(u, v) \in E(G)$ and $(u, v') \in E(G)$. Thus $N_G[u] \neq N_G[v]$ since $v' \in N_G[u]$ but $v' \notin N_G[v]$; i.e., u and v are not true twins. Moreover, the edge (u, v) does not have either endpoint in T. This is a contradiction to T being a twin cover and thus the claim is proved.

From the claim and the discussion prior to it, we have that $rtc(G) = \min\{a, b\}$.

Since a and b can be arbitrarily large, and $rtc(G) \geq tc(G)$, the restricted twin cover number of this class of graphs is unbounded. □

A bound on the pathwidth does not imply a bound on the restricted twin cover number, but a bound on the vertex cover number of a graph does.

Proposition 2.5.13. *There exist graph classes with bounded pathwidth but unbounded restricted twin cover number.*

Proof. Consider the class of paths on at least three vertices. Any path P_k for k 2 has pathwidth 1 but has vertex cover number at least $bk/2c$, and therefore twin cover number at least $bk/2c$. Therefore such a path has restricted twin cover number at least $bk/2c$, which may be arbitrarily large. □

Proposition 2.5.14. *For any graph G, $rtc(G)$ $vc(G)$; i.e., if $vc(G)$ is bounded then $rtc(G)$ is bounded.*

Proof. Let C be a minimum vertex cover of G, and let $k = jCj$ (so that $vc(G) = k$). Since G C is an independent set (as otherwise some edge does not have an endpoint in C, contradicting the definition of C), there are no non-trivial components of G C. Since every edge has an endpoint in C, C is also a twin cover.

Therefore, $rtc(G)$ $jCj = vc(G) = k$ as required. □

Figure 2.17 shows the position of the restricted twin cover number relative to other parameters.

Computing Restricted Twin Cover Number

In this section, we show how to compute $rtc(G)$ for a graph G in time $f(rtc(G))$ $n^{O(1)}$. To compute $rtc(G)$, we will use techniques from logic; specifically, we will appeal to Courcelle's theorem, which is stated later in this section (Theorem 2.5.18). This requires us to introduce *Monadic Second-Order* (MSO_1) logic[6]. Our presentation follows that of Obdržálek [116]. We start by defining MSO_1 which we can use to express properties of a graph.

Definition 2.5.15 (MSO_1 language)**.** *The language of MSO_1 on graphs contains the logical expressions that are built from the following elements:*

- *variables $x; y; x_1; : : : ; x_k$ for elements (vertices),*
- *variables $X; Y; X_1; : : : ; X_k$ for sets of vertices,*
- *the predicates $adj(x; y)$ (which is true if and only if $(x; y)$ 2 $E(G)$) and x 2 X,*

[6] We use MSO_1 to avoid confusion with MSO_2, another variant of the logic, which also allows quanti cation of sets of edges.

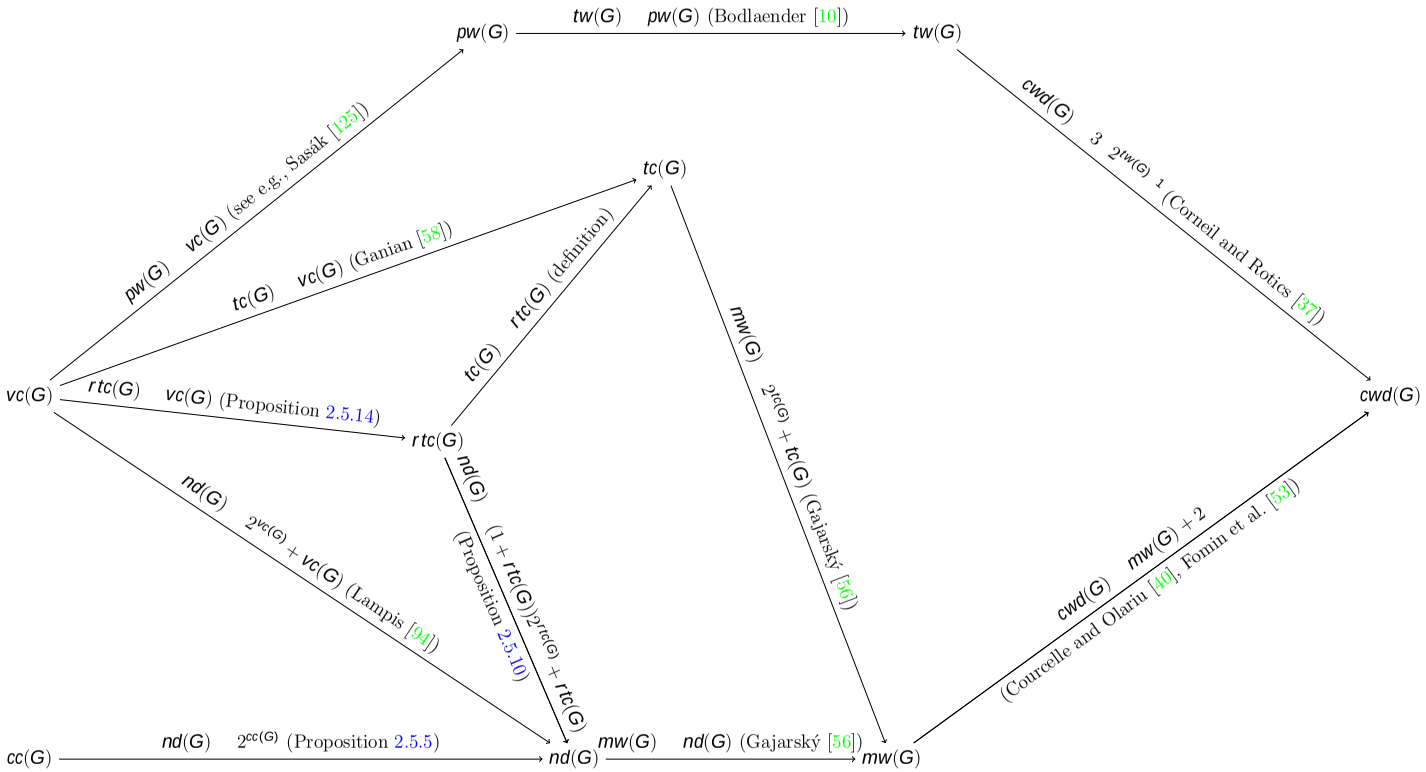


Figure 2.17: The relationship between restricted twin cover number and other parameters for graph problems.

- *equality of variables, the connectives* $\wedge, \vee, \neg, \rightarrow$,
- *vertex quantification* $\forall x, \exists x$, *and vertex set quantification* $\forall X, \exists X$.

Note that MSO_1 extends first-order logic by allowing quantification over sets of vertices.

Once we express a graph's properties in a logical formula, the next problem is to algorithmically check whether a graph satisfies a given formula. That is, given a graph G and a formula φ, we would like to determine whether $G \models \varphi$, that is, whether G "models" φ. This is the so-called *model checking* problem for the MSO_1 logic on graphs:

Problem 2.5.16. *(MSO_1* Model Checking*). Given an MSO_1 formula φ and a graph G, decide whether $G \models \varphi$.*

Note that the problem can be generalized for other logics and structures, but this is not relevant in this thesis. A more useful related problem is the following one, which encodes an optimization version of the previous problem. Consider any MSO_1 formula $\varphi(X_1, \ldots, X_p)$ with free set variables, and state the following problem on an input graph G:

$$opt\{f_{lin}(X_1, \ldots, X_p) : X_1, \ldots, X_p \subseteq V(G); G \models \varphi(X_1, \ldots, X_p)\};$$

where *opt* can be min or max, and f_{lin} is a linear evaluation function, which we define shortly. In this case, we want to find an assignment to the free set variables $X_1, \ldots, X_p$ that optimizes (either maximizes or minimizes) the function f_{lin} while maintaining that $G \models \varphi(X_1, \ldots, X_p)$ (i.e., such that G models the solution). The function f_{lin} can be seen as the combination of m functions f_i which can be seen as weights on the vertices of the graph. A linear evaluation function has the following form:

$$f_{lin}(X_1, \ldots, X_p) = \sum_{i=1}^{m} \sum_{j=1}^{p} a_{i,j} \sum_{x \in X_i} f_j(x) \tag{2.1}$$

where m and $a_{i,j}$ are (integer) constants and f_j are (integer) weight functions on the vertices of G. Often, f_{lin} is just the cardinality function. The optimization problem just defined is the following:

Problem 2.5.17. *($LinEMSO_1$) Given an MSO_1 formula φ, a linear evaluation function f_{lin}, and a graph G, compute an assignment z of free variables such that*

$$z(X_1, \ldots, X_p) = opt\{f_{lin}(X_1, \ldots, X_p) \mid G \models \varphi(X_1, \ldots, X_p)\};$$

i.e., one which is optimal.

We provide two examples of expressing graph properties in MSO_1. We will build upon these examples to determine a formula that can be used to compute the restricted twin cover number of the graph.

The Minimum Vertex Cover problem, which asks to compute the size of a smallest vertex cover in a graph G, can be expressed as:

$$(X) \quad 8u; v\ (adj(u; v)\ !\ (v\ 2\ X\ \_\ u\ 2\ X))$$

with $f_{lin}(X) = jXj$ so that the formula solves $\min(jXj)$ (for completeness, we can define this function as above by setting $m = p = a_{1;1} = 1$ and $f_1(x) = 1$ for all $x\ 2\ V(G)$). Similarly, the Minimum Twin Cover problem, which asks to compute the size of a smallest twin cover in a graph G, can be expressed as:

$$(X) \quad 8u; v\ (adj(u; v)\ !\ (v\ 2\ X\ \_\ u\ 2\ X\ \_\ twins(u; v)))$$

where "twins" means "true twins" in the formula and

$$twins(u; v) \quad adj(u; v) \wedge$$
$$8w \quad adj(u; w) \wedge adj(v; w)\ \_\ :adj(u; w) \wedge :adj(v; w) \quad ;$$

along with $f_{lin}(X) = jXj$ so that the the problem solves $\min(jXj)$.

Now we are ready to write a formulation for Minimum Restricted Twin Cover in MSO_1. The idea is to partition G into two sets: X_1 which will be a twin cover for which $rtc(G) = jTj + q$ for some $k \quad 0$, and X_2 which will be the set of representative vertices for the non-trivial components of $G \quad T$. We start with the following:

$$(X_1; X_2) \quad 8u; v \quad adj(u; v)\ ! \tag{2.2}$$
$$v\ 2\ X_1\ \_\ u\ 2\ X_1\ \_\ connected(u; v; X_1) \quad ;$$

where $connected(u; v)$ is true if and only if u and v are in some connected component C of $G \quad T$ and is defined as

$$connected(u; v; X_1) \quad 9C \quad (u\ 2\ C \wedge v\ 2\ C) \wedge \tag{2.3}$$
$$8x; y\ ((x\ 2\ C \wedge y\ 2\ C)\ !\ adj(x; y))\ \wedge \tag{2.4}$$
$$9w; z\ (w\ 2\ C \wedge w\ 2\ X_2 \wedge :(w\ 2\ X_1) \wedge z\ 2\ X_1 \wedge adj(w; z)) \tag{2.5}$$

along with $f_{lin}(X_1; X_2) = |X_1| + |X_2|$ so that the formula solves $\min(|X_1| + |X_2|)$.

Line 2.3 enforces that there is a set containing both u and v, while Line 2.4 enforces that this set is a complete graph. Line 2.5 enforces that there is a vertex w in the set which is in X_2 but not X_1: these vertices will be used count the number of components, and this vertex should be adjacent to a vertex in $X_1 = T$. Since all vertices of a component of $G \setminus T$ are true twins (as if they are not, there is an edge that is not covered by T which is not between true twins), the adjacency requirement to a vertex z in T ensures the graph is connected.

Finally, we note that one can define the linear evaluation function f_{lin} as in Equation 2.1 for the sum of the cardinalities of X_1 and X_2 as follows. We set $p = 2$ (necessarily, from the number of sets X_i given to f_{lin}), $m = 1$, $f_1(x) = 1$ for all $x \in V(G)$, and $a_{1,1} = a_{2,1} = 1$.

We can now use Courcelle's theorem, which we state next. In the statement, $|\varphi|$ denotes the length of the formula φ.

Theorem 2.5.18 (Theorem 4, Courcelle et al. [39])**.** *For every integer t and MSO_1 formula φ, every $LinEMSO_1$ optimization formula is fixed-parameter tractable on graphs of clique-width t, with the parameters t and $|\varphi|$.*

Theorem 2.5.19. *If $rtc(G) \leq k$, then we can find $T \subseteq V(G)$ such that $rtc(G) = |T| + q$ for some $q \geq 0$ in time $f(k) \cdot n^{O(1)}$. Therefore, computing $rtc(G)$ is fixed-pararameter tractable in the output-size.*

Proof. Suppose that $rtc(G) \leq k$ for some $k \geq 0$. By definition of $rtc(G)$, $tc(G) \leq rtc(G)$; thus G has a finite twin cover number. Gajarský [56] showed that $mw(G) \leq 2^{tc(G)} + tc(G)$; thus G has a finite modular width. It is known that $cwd(G) \leq mw(G) + 2$ (see, e.g., Courcelle and Olariu [40]); thus G has a bounded clique width.

We can therefore use Theorem 2.5.18 along with the formula φ (which has constant size) in Equation 2.2 to compute $T = X_1$ in time $f(k) \cdot n^{O(1)}$. □

Computing T used Courcelle's theorem (Theorem 2.5.18) which may not provide the most efficient algorithm. Therefore, while we know that for a graph with bounded restricted twin cover number a witness set T can be computed with an FPT algorithm, we do not necessarily have the most efficient algorithm to do so.

2.6 Search Algorithms

In this section, we define the graph search algorithms that we study for Type II problems in this thesis. We use the definitions of Corneil and Krueger [34].

A *graph search* algorithm is a systematic method for visiting all vertices in a graph. We now describe the algorithm *Generic Search*. Initially, all vertices of a graph are unnumbered. A graph search may start at any vertex and assigns the lowest number from $\{1, \ldots, n\}$ which has not been previously assigned to a vertex. We say that a search *visits* an unnumbered vertex v when it assigns it a number. The search then adds all unnumbered vertices of $N(v)$ to the set S of vertices to visit next. The search will then choose a vertex in S to visit next, and repeat the process above, adding unnumbered neighbours of the newly visited vertex to S. This process repeats until there are no more vertices to visit.

Recording vertices according to their assigned numbers naturally produces an ordering of the graph. The specific method by which unnumbered vertices are added to the set S, and the data structure used to store S, will affect the resulting order in which vertices are visited by the search.

Observe that since a search can only add vertices to S which are neighbours of visited vertices, a search only visits the component of the graph containing the initial vertex. For this reason, we only consider connected graphs for the S-End-Vertex problem: if the graph is not connected, one could search every other component in the graph before the component containing the target vertex.

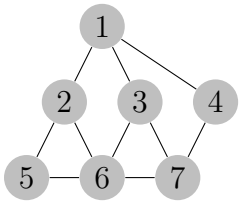
Recall that $L_i(z)$ denotes the ith layer of a graph with respect to z and is equal to all vertices at distance i from z, and that the eccentricity of a vertex, denoted ecc(v) is the largest distance between v and any other vertex in the graph. A search algorithm is a *layer search*, if, starting at an initial vertex u (which is assigned the number 1), the algorithm will visit each vertex in $L_i(u)$ $(0 \quad i < \ell)$ before any vertex of $L_j(u)$ for ecc(u) $\quad j > i$. The following is true for a layer search.

Observation 2.6.1. *Let be an ordering generated by a layer search. If* $d(\ (1); u) < d(\ (1); v)$ *for some vertices* $u; v \in V(G) \setminus \{\ (1)\}$, *then* $u < v$.

We will show results for the following search algorithms. An example execution of each search algorithm for an example graph is shown in Figure 2.18.

Breadth-First Search (BFS) is shown in Algorithm 2.6.1. BFS refines the description of Generic Search algorithm presented above by changing the set S to a *queue*[7]. BFS is a

[7] Some authors may call any algorithm that visits vertices in the graph according to the layers defined by the start vertex a breadth-first search.



BFS Ordering $= \langle 1, 2, 3, 4, 5, 6, 7 \rangle$

DFS Ordering $= \langle 1, 2, 5, 6, 7, 3, 4 \rangle$

LBFS Ordering $= \langle 1, 2, 3, 4, 6, 5, 7 \rangle$

MNS Ordering $= \langle 1, 2, 3, 4, 7, 6, 5 \rangle$

Figure 2.18: A graph and various example search orderings of it.

Algorithm 2.6.1: Breadth-First Search (BFS).

Input: $G = (V, E)$ and a start vertex $u \in V$.
Result: An ordering of the vertices of G.

```
1 initialize queue to {u};
2 for i ← 1 to |V| do
3 |   pop v from top of queue;
4 |   π(i) ← v; //This assigns v to the number i
5 |   for each unnumbered vertex w adjacent to v do
6 |   |   if w not already in queue then
7 |   |   |   push w on queue;
```

Algorithm 2.6.2: Lexicographic Breadth-First Search (LBFS).

Input: $G = (V, E)$ and a start vertex $u \in V$.
Result: An ordering of the vertices of G.

```
1 assign the label ⟨⟩ to all vertices;
2 label(u) ← ⟨|V| + 1⟩;
3 for i ← 1 to |V| do
4 |   pick any unnumbered vertex v with the largest lexicographic label;
5 |   π(i) ← v; //This assigns v to the number i
6 |   for each unnumbered vertex w in N(v) do
7 |   |   append (|V| − i) to label(w);
```

Algorithm 2.6.3: LBFS$^+$.

Input: $G = (V; E)$ and an ordering of V.
Result: An ordering of the vertices of G.

```
1 assign the label hi to all vertices;
2 for i  1 to jVj do
3 |  pick the unnumbered vertex v with the largest lexicographic label that is
  |   rightmost in ;
4 |   (i)  v; //This assigns v to the number i
5 |  for each unnumbered vertex w in N(v) do
6 |  |  append (jVj  i) to label(w);
```

layer search. BFS can be implemented to run in time $O(n+m)$ using adjacency lists (see, e.g., Bondy and Murty [14]).

Lexicographic Breadth-First Search (LBFS), Algorithm 2.6.2, is a refinement of BFS. LBFS essentially introduces a tie-breaking rule to BFS so that vertices on the same layer are chosen according the number and order of previously visited vertices in their neighbourhoods. LBFS achieves this by assigning labels, which are sequences (ordered sets), to unvisited vertices, and always choosing the lexicographically largest label when deciding which vertex to visit next. LBFS can be implemented to run in time $O(n+m)$ (see, e.g., Golumbic [65]). LBFS$^+$, Algorithm 2.6.3, is a variant of LBFS that refines the tie-breaking rule further. LBFS$^+$ additionally requires as input an ordering of the input graph, which it uses to break ties among lexicographically-largest labelled vertices, by taking the tied vertex which is rightmost in .

Depth-First Search (DFS), Algorithm 2.6.4, is Generic Search but uses a *stack* to store S. The result of changing the queue to a stack is that DFS is not a layer search. DFS can also be implemented to run in time $O(n+m)$, using adjacency lists (Tarjan [130]).

Maximal Neighbourhood Search (MNS), Algorithm 2.6.5, is a generalization of LBFS. Instead of the sequences used by LBFS as labels, MNS uses a set for labels. MNS uses an inclusion-wise maximal label as the next vertex to visit. As a result, MNS can "jump around" the graph and is not a layer search; a newly visited vertex need not be adjacent to the last vertex visited, nor be in the same layer. LBFS is a special case of MNS as the lexicographically largest label is always an inclusion-wise maximal label, but it may not be the only one. Li and Wu [98] show that MNS can be implemented to run in time $O(n+m)$.

Finally, we summarize the relationship between several search algorithms. Recall that LBFS refines BFS by adding a tie-breaking rule and that LBFS is also a refinement of MNS.

Algorithm 2.6.4: Depth-First Search (DFS).

Input: $G = (V; E)$, a start vertex $u \; 2 \; V$.
Result: An ordering of the vertices of G.

```
1 initialize stack to fug and set i  1;
2 while stack is not empty do
3     pop v from top of stack;
4     if v already has a number then
5         continue;
6      (i)  v; //This assigns v to the number i
7     i++;
8     for each unnumbered vertex w adjacent to v do
9         push w on top of stack;
```

Algorithm 2.6.5: Maximal Neighborhood Search (MNS).

Input: $G = (V; E)$, a start vertex $u \; 2 \; V$.
Result: An ordering of the vertices of G.

```
1 assign the label ; to all vertices;
2 label(u)  fjV j + 1g;
3 for i  1 to jV j do
4     pick any unnumbered vertex v with inclusion-wise maximal label;
5      (i)  v; //This assigns v to the number i
6     for each unnumbered vertex w in N(v) do
7         label(w)  label(w) [ fig;
```

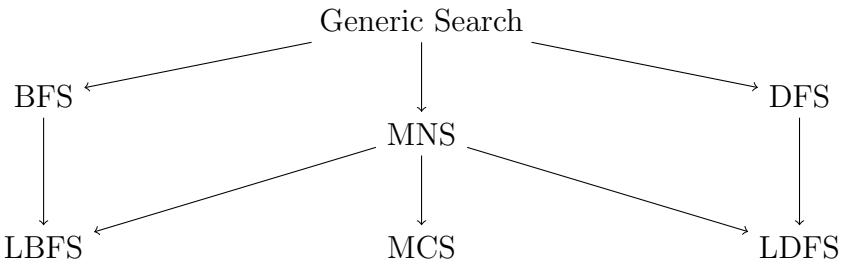


Figure 2.19: The relationship between search algorithms, as described in Corneil and Krueger [34]. An arrow $A \rightarrow B$ indicates that every ordering generated by algorithm B is also a valid ordering of A. For example, every LBFS-generated ordering is also a BFS-generated ordering as well as an MNS-generated ordering.

Figure 2.19 shows the relationship between the algorithms discussed in this section, as well as *Maximum Cardinality Search* (MCS) and *Lexicographic Depth-First Search* (LDFS). It is due to the relationship between these search algorithms that we obtain related orderings on graphs.

Part I

Type I Problems

Chapter 3

Cutwidth

3.1 Introduction

We start by listing some applications of Cutwidth, which helps to motivate the problem further and understand its importance. Among other applications, Cutwidth has use in

- circuit design and Very Large-Scale Integration (VLSI) (Breuer [19], Makedon and Sudborough [108], Adolphson and Hu [1], Adolphson [2], Chung et al. [28], Lagergren [97], Fellows and Langston [51], Raspaud et al. [122]),
- network reliability (Karger [88]),
- graph drawing (Mutzel [114], Shahrokhi et al. [126]),
- heuristics for SAT-solving (Wang et al. [136]),
- protein similarity detection (Blin et al. [9]),
- information retrieval (Botafogo [15]), and
- as a subroutine for the traveling salesman problem (Junger et al. [84]).

We now turn to the complexity of the problem. In general, Cutwidth is NP-complete (Gavril [62]). Monien and Sudborough [113] showed that Cutwidth is NP-complete on planar graphs with $\Delta(G)$ 3. Diaz et al. [43] showed that the problem remains NP-complete on (partial) grids (and therefore bipartite graphs) and unit disk graphs. More

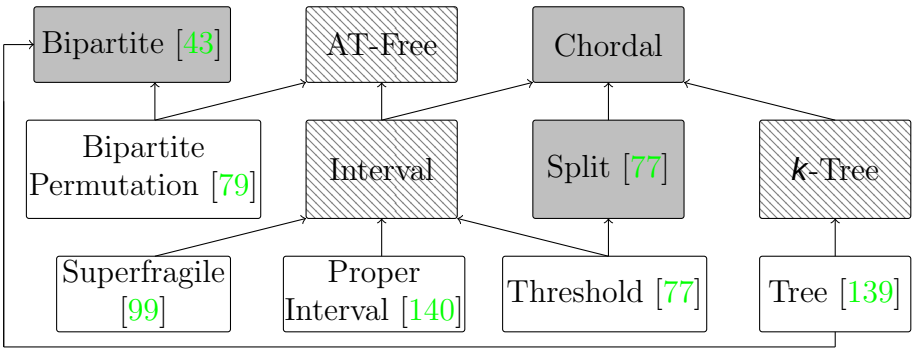
recently, Heggernes et al. [77] showed that Cutwidth is NP-complete for split graphs (even when all vertices v in the independent set have $d(v) = 2$); consequently for all graph classes containing split graphs, e.g., chordal graphs, the problem is NP-complete.

On restricted graph classes, there are positive results. On very small graph classes, closed formulas for the cutwidth of graphs are known (Harper [75], Lin et al. [100], Lin et al. [101], Nakano [115]). Yannakakis [139] showed that Cutwidth is solvable in polynomial time on trees (and that a version of Cutwidth which does not allow edge crossings has a linear time solution on trees). Rolim et al. [123] showed $O(n^2)$ algorithms for some classes of mesh graphs. Heggernes et al. [77] showed that Cutwidth has a linear time solution on threshold graphs. Heggernes et al. [79] later showed that Cutwidth has a linear time solution on bipartite permutation graphs. Yuan and Zhou [140] showed that left-endpoint orderings of proper interval graphs are also cutwidth-minimal orderings, and since these can be found in linear time, Cutwidth has a linear time solution on these graphs. Lilleeng [99] showed that Cutwidth has a polynomial-time solution for superfragile graphs.

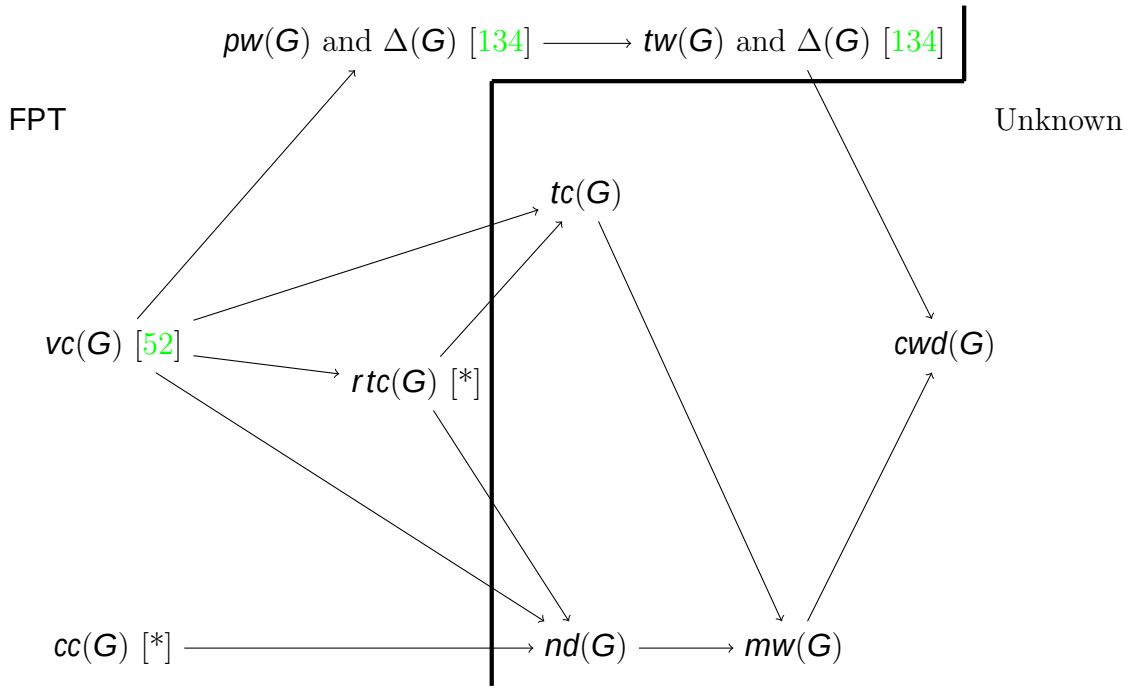
Some of the above complexity results for Cutwidth are shown in Figure 3.1a. Many graph classes for which Cutwidth is tractable are subsets of AT-free graphs. Notably, the complexity of Cutwidth on interval graphs, the intersection of AT-free and chordal graphs, is an open problem. Even for cographs—graphs with no induced P_4—the problem remains open.

There are also some approximation results for Cutwidth. Cutwidth can be approximated to within $O(\log^2 n)$ on general graphs in polynomial time (Leighton and Rao [95]). Cutwidth can be approximated to within a constant factor on dense graphs in polynomial time (Arora et al. [3]).

There are positive results for Cutwidth in the parameterized complexity setting. Cutwidth is FPT when parameterized by the size of the solution (Thilikos et al. [133]), and Giannopoulou et al. [63] provided a simpler algorithm for the same parameter. Fellows et al. [52] showed that Cutwidth is FPT when parameterized by the size of the minimum vertex cover of the graph $vc(G)$, with an algorithm which runs in time $O(2^{2^{O(vc(G))}} n^{O(1)})$. Fellows et al. [52] were not concerned with optimizing the run time, and Cygan et al. [42] later showed an improved algorithm running in time $O(2^{vc(G)} n^{O(1)})$. Cygan et al. [42] also showed that Cutwidth parameterized by the vertex cover number does not admit a polynomial kernel unless NP coNP=poly. Bodlaender et al. [11] showed that Cutwidth parameterized by the solution size is unlikely to have a polynomial-sized kernel. Cutwidth is NP-complete for treewidth 2 and pathwidth 3 graphs (Monien and Sudborough [113]), but Thilikos et al. [134] show that Cutwidth is FPT when parameterized by the treewidth (or pathwidth) of the graph and the maximum degree. Some known



(a) Some known complexity results for Cutwidth on restricted graph classes. The problem is NP-complete for classes with a solid gray background, has unknown complexity for classes with a hatched background, and is in P otherwise.



(b) Some known parameterized complexity results for Cutwidth for common graph parameters. For each parameter to the left of the thick line, there is an FPT algorithm for Cutwidth with that parameter. The complexity of Cutwidth is open for those parameters to the right of the thick line. FPT algorithms for parameters marked with [*] are shown in this work.

Figure 3.1: Some known complexity results for Cutwidth.

fixed-parameter complexity results for Cutwidth are shown in Figure 3.1b.

The lack of understanding of cutwidth-minimal orderings on cographs may be the reason why these problems are not very well understood in the parameterized complexity setting. For graphs of bounded clique width, fixed-parameter tractable algorithms for these problems are not known. Since cographs have clique width at most 2 (Courcelle and Olariu [40]), a hardness result on cographs will translate into a hardness result for graphs of bounded clique width, while an FPT algorithm would mean a polynomial time solution for these problems on cographs. In fact, in the parameterized setting, this problem has no known hardness results. Cutwidth is fixed-parameter tractable for graphs with a bounded vertex cover number, but its complexity on more general parameters is unknown, except for small generalizations or with the aid of additional parameters.

Finally, Cutwidth is related to other graph parameters. For example, it is at least as large as a graph's so-called *topological bandwidth* (Chung and Seymour [27], Chung [25], Makedon et al. [107]). The cutwidth of a graph is also related to a graph's pathwidth and treewidth (see, e.g., Korach and Solel [90]). Lokshtanov et al. [104] showed the following lemma, which summarizes these relationships.

Lemma 3.1.1 (Lemma 2, Lokshtanov et al. [104])**.** *Let G be a graph. Then:*

$$tw(G) \quad pw(G) \quad cw(G) \quad \frac{im(G)}{2}:$$

Furthermore, $\Delta(G) \quad im(G)$.

Moreover, Bodlaender et al. [12] showed that there is a common algorithm that can be used as a starting point to compute the cutwidth or pathwidth of a graph, and explore weighted and directed versions of these parameters.

Summary of Results

In this chapter, we first show some results on small graph classes (Section 3.3) and establish some connections to Imbalance (Section 3.4). We then establish a new theorem which shows that for any graph, there is a cutwidth-minimal ordering where the vertices of each equivalence class of true twins appear consecutively (Section 3.5). Using this theorem, we extend the result of Fellows et al. [52] to graphs with bounded restricted twin cover number (Section 3.7) and show that Cutwidth is FPT when the parameter is the edge clique cover number of a graph (Section 3.6).

3.2 Preliminaries

In this section we present some useful facts related to Cutwidth. Recall that the rank of a vertex with respect to an ordering is rank $(v) = \text{succ}\ (v)$ pred (v), where succ (v) is the number of v's neighbours to its right in and pred (v) is the number of v's neighbours to its left in .

First, we note some observations related to the exchange of two consecutive vertices in an ordering .

Observation 3.2.1. *Let G be a graph and be an ordering of G where $x <$ y and $fx;yg$ is consecutive in . If $(x;y) \in E(G)$ and is obtained by swapping the positions of x and y, then rank $(x) =$ rank (x) and rank $(y) =$ rank (y).*

Lemma 3.2.2. *Let G be a graph and be an ordering of G where $x <$ y and $fx;yg$ is consecutive in . If $(x;y) \in E(G)$ and is obtained by swapping the positions of x and y, then rank $(x) =$ rank (x) 2 and rank $(y) =$ rank $(y) + 2$.*

Proof. First, observe that succ $(x) =$ succ $(x)+1$ as all the successors of x in other than y are still successors of x in , but y is also a successor of x in . Similarly, succ $(y) =$ succ (y) 1 as all the successors of y in are still successors of y in , except for x. Next, pred $(x) =$ pred (x) 1 as all the predecessors of x in are still predecessors of x in , except for y. Similarly, pred $(y) =$ pred $(y) + 1$ as all the predecessors of y in are still predecessors of y in , but x is also a predecessor of y in .

Putting the above facts to work, we can explicitly compute the changes to the ranks. Observe the change to the rank of x:

$$\begin{aligned}\text{rank}\ (x) &= \text{succ}\ (x) \quad \text{pred}\ (x) \\ &= (\text{succ}\ (x) + 1) \quad (\text{pred}\ (x) \quad 1) \\ &= \text{succ}\ (x) \quad \text{pred}\ (x) + 2 \\ &= \text{rank}\ (x) + 2;\end{aligned}$$

Finally, we can compute the change to the rank of y:

$$\begin{aligned}\text{rank}\ (y) &= \text{succ}\ (y) \quad \text{pred}\ (y) \\ &= (\text{succ}\ (y) \quad 1) \quad (\text{pred}\ (y) + 1) \\ &= \text{succ}\ (y) \quad \text{pred}\ (y) \quad 2 \\ &= \text{rank}\ (y) \quad 2;\end{aligned}$$

as required. □

Recall that moving a vertex (i) forward to a position j (for some $j \geq i$) in an ordering results in an ordering where (i) has been moved right to be $'(j)$ but leaves the relative ordering of the other vertices unchanged. Moving a vertex backward is analogous, but the vertex is moved left. Moving a vertex forward means that $j \geq i$, while moving a vertex backwards means that $j \leq i$. These follow from the fact that moving a vertex forward can only decrease the rank of the vertex by 2, and moving a vertex backwards can only increase the rank of the vertex by 2 (Lemma 3.2.2).

Observation 3.2.3. *Let be an ordering of a graph and (i) be a vertex of even degree. If rank $(\ (i)) < 0$ and moving (i) to position j results in an ordering $'$ such that rank $'(\ (i)) > 0$, then there is a position between i and j in where (i) would have rank exactly zero.*

Observation 3.2.4. *Let be an ordering of a graph and (i) be a vertex of odd degree. If rank $(\ (i)) < 1$ and moving (i) to position j results in an ordering $'$ such that rank $'(\ (i)) > 0$, then there is a position between i and j in where (i) would have rank exactly 1. Similarly, if rank $(\ (i)) > 1$ and moving (i) to position j results in an ordering $'$ such that rank $'(\ (i)) < 1$, then there is a position between i and j in where (i) would have rank exactly 1.*

Next, we list some observations regarding the reversal of an ordering.

Observation 3.2.5. *If is an ordering of a graph with its reverse ordering ^{R}, then $cw(\ ) = cw(\ ^R)$.*

Observation 3.2.6. *If is an ordering of a graph with its reverse ordering ^{R}, then rank $(v) =$ rank $_R(v)$ for all vertices v of .*

Recall that $c\ (v)$ is the size of the cut after vertex v with respect to . The following was observed by Lokshtanov et al. [104].

Observation 3.2.7. *For any ordering and $1 \leq j \leq n - 1$, $c\ (\ (j+1)) = c\ (\ (j)) +$ rank $(\ (j+1))$. Therefore, $c\ (\ (j+1)) = \sum_{i=1}^{j+1}$ rank $(\ (i))$.*

Observation 3.2.8. *If an ordering of a graph is such that rank $(\ (i)) \geq 0$ for all $1 \leq i \leq k$, and rank $(\ (j)) \leq 0$ for all $k+1 \leq j \leq n$, then*

$$cw(\ ) = \max_{1 \leq k' \leq n} \sum_{i=1}^{k'} \text{rank } (\ (i)) = cw(\ (k)).$$

As we now show, the next lemma establishes a relationship between the ranks of twins in an ordering. Recall that false twins have the same open neighbourhood, and true twins have the same closed neighbourhood.

Lemma 3.2.9. *Let u and v be twins with $u < v$ in an ordering . If u and v are false twins, then rank (u) rank (v). If u and v are true twins, then rank (u) rank $(v)+2$.*

Proof. First, suppose that u and v are false twins, i.e., $N(u) = N(v)$. Therefore succ (u) succ (v) since every neighbour of v to the right of v in is also to the right of u and any vertices of $N(u) = N(v)$ between u and v are also to the right of u. Similarly, pred (u) pred (v) since every neighbour of u to the left of u in is also to the left of v and any vertices of $N(u) = N(v)$ between u and v are also to the left of v. Moreover, for any vertex x and any ordering , succ (x) 0 and succ (x) 0. Therefore we have

$$\begin{aligned}
\text{rank } (u) &= \text{succ } (u) \quad \text{pred } (u) \\
&\quad \text{succ } (u) \quad \text{pred } (v) \text{ (since pred } (v) \quad \text{pred } (u) \quad 0) \\
&\quad \text{succ } (v) \quad \text{pred } (v) \text{ (since } 0 \quad \text{succ } (v) \quad \text{succ } (u)) \\
&= \text{rank } (v);
\end{aligned}$$

as required.

Now suppose that u and v are true twins, i.e., $N[u] = N[v]$. Therefore succ (u) 1 succ (v), or equivalently succ (u) succ $(v) + 1$ since every neighbour of v to the right of v in is also a neighbour to the right of u, and vertex v is a neighbour of u to the right by $(u; v)$ 2 $E(G)$. Similarly, pred (u) pred (v) 1 since every neighbour of u to the left of u in is also a neighbour to the left of v, and u is a neighbour of v to the left by $(u; v)$ 2 $E(G)$. Moreover, for any vertex x and any ordering , succ (x) 0 and succ (x) 0. Therefore we have

$$\begin{aligned}
\text{rank } (u) &= \text{succ } (u) \quad \text{pred } (u) \\
&\quad \text{succ } (u) \quad (\text{pred } (v) \quad 1) \text{ (since pred } (v) \quad 1 \quad \text{pred } (u) \quad 0) \\
&\quad \text{succ } (v) + 1 \quad (\text{pred } (v) \quad 1) \text{ (since } 0 \quad \text{succ } (v) + 1 \quad \text{succ } (u)) \\
&= \text{rank } (v) + 2;
\end{aligned}$$

as required. □

The above lemma leads to the following observation.

Observation 3.2.10. *If X is a set of twins in some graph, then the ranks of the vertices of X are non-increasing in any ordering of the graph containing X.*

The following lemma allows us to rearrange some vertices in an ordering without increasing the cutwidth of the ordering.

Lemma 3.2.11 (Lemma 1, Cygan et al. [42]). *If moving $v_p = \ (p)$ backward to a position q results in an ordering $^{\prime}$ such that rank $_{\prime}(v_p) \quad 0$, then $cw(\ ^{\prime}) \quad cw(\ )$. If moving $v_p = \ (p)$ forward to a position q results in an ordering $^{\prime}$ such that rank $_{\prime}(v_p) \quad 0$, then $cw(\ ^{\prime}) \quad cw(\ )$.*

The lemma above can be applied to move entire sets of twins in some scenarios. Recall that $_A$ denotes the ordering of A imposed by .

Lemma 3.2.12. *Let $G = (V; E)$ be a graph with an ordering , $A \quad V$ be a set of twins, and $_A = ha_1; : : : ; a_m i$.*

1. *If rank $(a_j) \quad 0$ for some $1 \quad j \quad m$, then for any $1 \quad i \quad j$, there is an ordering $^{\prime}$ of G such that a_j has not been moved (i.e., $(a_j) = \ ^{\prime}(a_j)$), all vertices $fa_i; a_{i+1}; : : : ; a_j g$ are consecutive, $^{\prime}_A = \ _A$, $^{\prime}_{V nA} = \ _{V nA}$, and $cw(\ ^{\prime}) \quad cw(\ )$.*
2. *If rank $(a_j) \quad 0$ for some $1 \quad j \quad m$, then for any $j \quad i \quad m$, there is an ordering $^{\prime}$ of G such that a_j has not been moved (i.e., $(a_j) = \ ^{\prime}(a_j)$), all vertices $fa_j; a_{j+1}; : : : ; a_m g$ are consecutive, $^{\prime}_A = \ _A$, $^{\prime}_{V nA} = \ _{V nA}$, and $cw(\ ^{\prime}) \quad cw(\ )$.*

Proof. We prove statement (1) only, as statement (2) follows from Observations 3.2.6 and 3.2.7 and statement (1).

The proof is by induction on $j \quad i$. If $i = j$, then setting $^{\prime} = \ $ gives the claim. Assume the result holds for $j \quad i = k$ for some $k \quad 0$ and consider $j \quad i = k + 1$.

By the induction hypothesis, there is an ordering $^{\prime}$ of G such that $fa_{i+1}; : : : ; a_j g$ are consecutive, a_j has not moved, $^{\prime}_A = \ _A$, $^{\prime}_{V nA} = \ _{V nA}$, and $cw(\ ^{\prime}) \quad cw(\ )$. Since a_j has not moved, and since $^{\prime}_A = \ _A$ and $^{\prime}_{V nA} = \ _{V nA}$, the predecessors and successors of a_j cannot change. Therefore, rank $_{\prime}(a_j) =$ rank (a_j); since rank $(a_j) \quad 0$, rank $_{\prime}(a_j) \quad 0$ too. Since A is a set of twins, by Observation 3.2.10, $a_1; : : : ; a_j$ must all have non-negative ranks. Therefore, rank $_{\prime}(a_i) \quad 0$.

Let $^{\prime\prime}$ be the result of moving a_i forward in $^{\prime}$ so that it is immediately to the left of a_{i+1}. This operation does not affect the rank of a_{i+1}, which was non-negative by Observation 3.2.10 and rank $_{\prime}(a_i) \quad 0$. Since a_i and a_{i+1} are twins, rank $_{\prime\prime}(a_i) \quad$ rank $_{\prime\prime}(a_{i+1})$. By Lemma 3.2.11 and the fact that rank $_{\prime\prime}(a_i) \quad 0$, we have that $cw(\ ^{\prime\prime}) \quad cw(\ ^{\prime})$. By the induction hypothesis, $cw(\ ^{\prime}) \quad cw(\ )$, and so $cw(\ ^{\prime\prime}) \quad cw(\ )$ as required.

Finally, note that $^{\prime\prime}_{V nA} = \ _{V nA}$ since no vertices of $V \ n \ A$ have been re-ordered. Since no vertex of A has moved past another vertex of A, we also have that $^{\prime\prime}_A = \ _A$. □

3.3 Small Graph Classes

In this section, we provide explicit formulas for the cutwidth of complete graphs and complete bipartite graphs. These enable us to derived explicit formulas for the imbalance of these graphs in Section 4.3. We provide an explicit formula for the cutwidth of a complete graph first.

Lemma 3.3.1. *If G is a complete graph on n vertices, then $cw(G) = \frac{n}{2}\ \frac{n}{2}$. Moreover, any ordering of K_n is optimal, $c(\ (bn/2c))$ is the (first) largest cut, rank$(\ (i)) \quad 0$ for all $1 \quad i \quad bn/2c$, and rank$(\ (j)) \quad 0$ for all $bn/2c + 1 \quad j \quad n$.*

Proof. Since all vertices of a complete graph are true twins, every ordering of K_n has $cw(\ ) = cw(G)$. Observe that

$$\begin{aligned}\text{rank}(\ (i)) &= \text{succ}(\ (i)) \quad \text{pred}(\ (i)) \\ &= (n \quad i) \quad (i \quad 1) \\ &= n \quad 2i + 1;\end{aligned}$$

which is positive if and only if $i < (n+1)/2$, i.e., $i \quad bn/2c$. By Observation 3.2.8,

$$cw(\ ) = \max_{1 \quad k' \quad n} \sum_{i=1}^{k'} \text{rank}(\ (i)) = \sum_{i=1}^{bn/2c} \text{rank}(\ (i)) = cw(\ (bn/2c)).$$

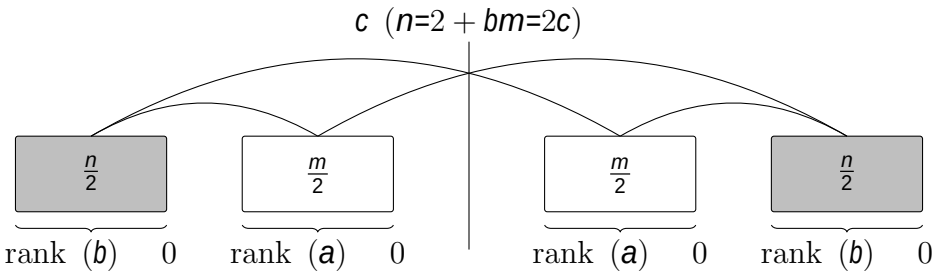
Therefore the largest cut in such an ordering is immediately after $\frac{n}{2}$ vertices, where each of the first $\frac{n}{2}$ vertices have an edge to the remaining $\frac{n}{2}$ vertices; thus $\frac{n}{2}\ \frac{n}{2}$ edges cross this cut, and $cw(\ ) = \frac{n}{2}\ \frac{n}{2}$. □

The next lemma is a closed formula for the cutwidth of a complete bipartite graph. An illustration of the orderings provided by Lemma 3.3.2 are shown in Figure 3.2.

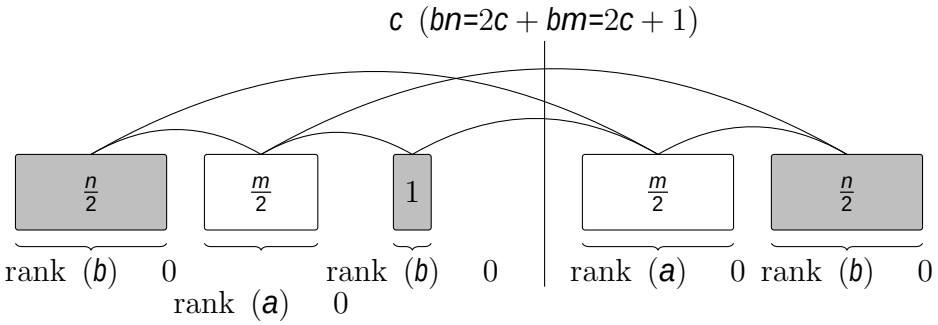
Lemma 3.3.2. *Consider a complete bipartite graph $K_{m,n}$. Then*

$$cw(K_{m,n}) = \begin{cases} \frac{mn}{2} & \text{if } mn \text{ is even} \\ \frac{mn+1}{2} & \text{if } mn \text{ is odd} \end{cases}$$

Moreover, there is a cutwidth-minimal ordering of $K_{m,n}$ such that there is a number $1 \quad k \quad m+n$ where k is the index of a largest cut, rank$(\ (i)) \quad 0$ for all $1 \quad i \quad k$, and rank$(\ (j)) \quad 0$ for all $k+1 \quad j \quad m+n$.



(a) The ordering constructed by Lemma 3.3.2 when mn is even. We assume without loss of generality that jBj is even.



(b) The ordering constructed by Lemma 3.3.2 when mn is odd.

Figure 3.2: Structure of cutwidth-minimal orderings for $K_{m;n}$ constructed by Lemma 3.3.2. In each case, we consider a complete bipartite graph $G = (A; B; E)$ where $jAj = m$ and $jBj = n$. Each box represents an arbitrary ordering of the specified size of vertices from A if it is white or of vertices from B if it is gray. An edge between boxes indicates that every vertex in one box is adjacent to every vertex in the other box. The ordering within a box is not relevant since all vertices in the same partite set are false twins. The vertical line indicates the position of the first largest cut in each ordering.

Proof. Let $G = (A; B; E)$ be a complete bipartite graph $K_{m;n}$ where $jAj = m$ and $jBj = n$.

We want to find a cutwidth-minimal ordering of G such that if $_A = ha_1; : : : ; a_m i$, then rank (a_i) 0 for some a_i 2 A. To do so, start with an arbitrary cutwidth-minimal ordering $\hat{}$ of G, and observe that

- if rank$_\wedge(a_1)$ 0, then using $= \hat{}$ and $i = 1$ gives the desired ordering and index,
- else if rank$_\wedge(a_m)$ 0, by Observations 3.2.5 and 3.2.6, we can take $= (\hat{})^R$ and use $i = 1$,
- otherwise rank$_\wedge(a_1) < 0$ and rank$_\wedge(a_m) > 0$. However, since $a_1 <_\wedge a_m$, this case is impossible, as it would contradict Lemma 3.2.9.

Therefore, a cutwidth-minimal ordering always exists such that some vertex a_i has non-negative rank. Among all possible choices for , choose one which maximizes the index i for a_i. By Observation 3.2.10, for all a_j 2 A with 1 $j < i$, rank (a_j) rank (a_{j+1}).

Case 1: n is even. By Lemma 3.2.9 for 1 $j < i$, rank (a_j) rank (a_{j+1}).

Since $d(a_i) = n$ is even, we now claim that we can move a_i forwards until rank $(a_i) = 0$. Since a_i was chosen so that i was maximal, it must have been that rank $(a_{i+1}) < 0$ as otherwise we would have chosen a_{i+1}, which is further right than a_i. By Observation 3.2.3, since $N(a_i) = N(a_{i+1})$ and rank $(a_{i+1}) < 0$, moving a_i to be immediately left of a_{i+1} would result in the ranks of a_i and a_{i+1} being equal, and there must be a spot to the right of a_i before this where a_i would be perfectly balanced. By Lemma 3.2.11, we can move a_i forwards until the rank of a_i is 0 in the resulting ordering.

By Lemma 3.2.12, there is a cutwidth-minimal ordering ^{0} such that all vertices of $fa_1; a_2; : : : ; a_i g$ are consecutive, rank $_0(a_1)$ rank $_0(a_2)$: : : rank $_0(a_i)$ 0, $^0_B = _B$, and $^0_A = _A$. Note that as A is a set of false twins, rank $_0(a_j) =$ rank (a_j) for all 1 j i.

We claim that for all a_j, $i < j$ m, 0 rank $_0(a_{j\ 1})$ rank $_0(a_j)$. First, note that since rank $_0(a_i) = 0$, by Lemma 3.2.9, 0 rank $_0(a_j)$ for $i < j$ m. The fact that rank $_0(a_{j\ 1})$ rank $_0(a_j)$ for all $i < j$ m is also immediate from Lemma 3.2.9. Therefore, we are able to apply Lemma 3.2.12 to ^{0} using the index of a_i get a cutwidth-minimal ordering where all vertices of $fa_i; : : : ; a_m g$ appear consecutively; call the resulting ordering . Since $fa_1; : : : ; a_i g$ also appear consecutively, the set A is consecutive in .

So we have that $= hB_1$ A $B_2 i$ for some sets $B_1; B_2$ which partition B. Since any vertex of b only has neighbours in A, in , all vertices of B_1 have positive rank equal to $jAj = m$, while vertices of B_2 have negative rank equal to $jAj =$ m.

Since all vertices of A were moved to a position where their rank is equal to that of the rank of a_i, and rank $(a_i) = 0$, all vertices of A have rank 0. Since

$$\begin{aligned} 0 = \text{rank}\ (a_i) &= \text{succ}\ (a_i) \quad \text{pred}\ (a_i) \\ &= jB_2j \quad jB_1j; \end{aligned}$$

we must have that $jB_2j = jB_1j$.

Call the first $dm=2e$ vertices of A the set A_1 and let $A_2 = A \setminus A_1$. The vertices of A_1 have non-negative rank (equal to zero), while the vertices of A_2 have non-positive rank (also equal to zero). This ordering satisfies $cw(\ ) = cw(G)$, as well as the second part of the lemma, as we can take $k = n=2 + bm=2c$. It remains to be shown that $cw(\ ) = mn=2$.

By Observation 3.2.8, the maximum cut of is after the first $dm=2e$ vertices of A and the first $jB_1j = n=2$ vertices of B. Recall that $E(X;Y)$ denotes the edges with one endpoint in X and the other in Y. The value of this cut is therefore

$$\begin{aligned} cw(G) = cw(\ ) &= jE(B_1;A_2)j + jE(B_2;A_1)j \\ &= \frac{n}{2}\ \left\lfloor \frac{m}{2} \right\rfloor + \frac{n}{2}\ \left\lceil \frac{m}{2} \right\rceil = \frac{n}{2}\ \left\lfloor \frac{m}{2} \right\rfloor + \left\lceil \frac{m}{2} \right\rceil = \frac{mn}{2}; \end{aligned}$$

as required by the lemma, since mn is always even if n is even. Thus, the lemma holds in the case that n is even.

Case 2: n is odd. If m is even, then nm is even, and we are done by the case above (since we can relabel the partite sets). So we may assume that m is odd as well.

Let rank (a_i) 0 for some a_i 2 A, and choose a_i to be rightmost in among all vertices that satisfies these constraints.

We claim that can move a_i forward until rank $(a_i) = 1$. First we show that rank (a_{i+1}) 1 (note that rank (a_{i+1}) 6= 0 as $d_G(a_i)$ is odd for any ordering). Since G is a complete bipartite graph, $N(a_i) = N(a_{i+1})$, and by Lemma 3.2.9, rank (a_i) rank (a_{i+1}). Therefore, rank (a_{i+1}) < 0 by choice of i. By Observation 3.2.4, since $N(a_i) = N(a_{i+1})$ and rank (a_{i+1}) < 0, moving a_i to be immediately left of a_{i+1} would result in the ranks of a_i and a_{i+1} being equal, and there must be a spot to the right of a_i before this where a_i would be perfectly balanced. By Lemma 3.2.11, we can move a_i forwards until the rank of a_i is 1 in the resulting ordering.

By Lemma 3.2.12, there is a cutwidth-minimal ordering ^{0} such that all vertices of $fa_1;a_2;\dots;a_ig$ are consecutive, $^0_B = \ _B$, and $^0_A = \ _A$. Note that as A is a set of false twins, rank $_0(a_j) =$ rank (a_i) for all 1 j i.

By choice of a_i and the fact that n is odd, by Lemma 3.2.9, we must in fact have that 1 rank $_{\emptyset}(a_j)$ for $i < j$ m. By Observation 3.2.4, since $N(a_i) = N(a_{i+1})$ and rank $(a_{i+1}) < 0$, moving a_{i+1} to be immediately right of a_i would result in the ranks of a_i and a_{i+1} being equal, and there must be a spot to the right of a_i before this where a_{i+1} would be perfectly balanced, with rank equal to 1. By Lemma 3.2.11, we can move a_{i+1} backwards until the rank of a_{i+1} is 1 in the resulting ordering.

Since a_i has rank 1 and a_{i+1} has rank 1, and they are false twins, there must be a single vertex b 2 B between a_i and a_{i+1}. To see this, note that since rank $_{\emptyset}(a_{i+1}) = 1$, there is one more vertex of B to its left than its right; necessarily we must have $bn=2c$ vertices of B on its right. Since rank $_{\emptyset}(a_i) = 1$, there is one more vertex of B to its right than its left; necessarily we must have $bn=2c$ vertices of B on its left. Therefore, only one vertex remains in B to be placed between a_i and a_{i+1}.

Thus, we are able to apply Lemma 3.2.12 to $^{\emptyset}$ using the index of a_{i+1} get a cutwidth-minimal ordering where the vertices of A appear in two maximal consecutive groups split by a single vertex from B. Call the resulting ordering ^{y}.

We have therefore established that $^{y} = hB_1$ A_1 B_2 A_2 B_3i for some sets $A_1; A_2$ which partition A and sets $B_1; B_2; B_3$ which partition B. As established above, $jB_2j = 1$ and $jB_1j = jB_3j = bn=2c$. If jA_1j $bm=2c$, set $=$ ^{y}. Otherwise, we have that $jA_1j < bm=2c$, and we can set to the reverse of ^{y} (i.e., $= (\ ^{y})^{R}$). Either way, jA_1j $bm=2c$ in .

Let $B_2 = fbg$. If $jA_1j > bm=2c$, then rank $(b) <$ 1: b has strictly more than half of its neighbourhood $N(b) = A$ to its left, so it has strictly less than half of its neighbourhood to its right, and therefore must have a negative rank by definition. By Observation 3.2.4, since $N(b) = A$ and rank $(b) <$ 1, there must be a spot to the left of b where b would be perfectly balanced, with rank equal to 1. Using Lemma 3.2.11, we can move b backwards until it has rank exactly 1; call the resulting order .

There are exactly $bm=2c$ vertices of A after b in . In particular, $= hB_1$ $A_1^{\emptyset}$ B_2 $A_2^{\emptyset}$ B_3i where $jA_2^{\emptyset}j = bm=2c$ (and therefore $jA_1^{\emptyset}j = dm=2e$). Let $=$ ^{R}. We can write $=$ $hB_1^{\emptyset\emptyset}$ $A_1^{\emptyset\emptyset}$ fbg $A_2^{\emptyset\emptyset}$ $B_3^{\emptyset\emptyset}i$ where $jB_1^{\emptyset\emptyset}j = jB_3^{\emptyset\emptyset}j = bn=2c$, $jB_2^{\emptyset\emptyset}j = 1$, $jA_1^{\emptyset\emptyset}j = bm=2c$, and $jA_1^{\emptyset\emptyset}j = dm=2e$.

In , all vertices of $B_1^{\emptyset\emptyset}$, $A_1^{\emptyset\emptyset}$, along with b, have positive rank. All vertices of $B_3^{\emptyset\emptyset}$ and $A_2^{\emptyset\emptyset}$ have negative rank. This ordering satisfies $cw(\ ) = cw(G)$, since all operations can only decrease the cutwidth. Hence, $cw(\ ) = cw(G)$ since was a cutwidth-minimal ordering of G.

By Observation 3.2.8, the maximum cut of is after $bn=2c + bm=2c + 1$ vertices. Thus, this ordering satisfies second part of the lemma. The value of the cut after $bn=2c+bm=2c+1$

vertices is therefore

$$\begin{aligned} cw(G) = cw(\ ) &= jE(A_1^{00}; B_3^{00})j + jE(fbg; A_2^{00})j + jE(A_2^{00}; B_1^{00})j \\ &= \left\lfloor \frac{m}{2} \right\rfloor \left\lfloor \frac{n}{2} \right\rfloor + \left\lfloor \frac{m}{2} \right\rfloor \left\lfloor \frac{m}{2} \right\rfloor + \left\lfloor \frac{n}{2} \right\rfloor \left\lfloor \frac{m}{2} \right\rfloor \\ &= \left\lfloor \frac{n}{2} \right\rfloor \left\lfloor \frac{m}{2} \right\rfloor + \left\lfloor \frac{m}{2} \right\rfloor^2 + \left\lfloor \frac{n}{2} \right\rfloor \left\lfloor \frac{m}{2} \right\rfloor \\ &= \left\lfloor \frac{n}{2} \right\rfloor (m) + \frac{m+1}{2} = \left(\frac{n}{2} - \frac{1}{2} \right) (m) + \frac{m+1}{2} \\ &= \frac{nm+1}{2}; \end{aligned}$$

as required by the lemma, since mn is odd. Thus, the lemma holds in the case that n is odd. □

3.4 Connections to Imbalance

In this section, we collect some results that connect Cutwidth and Imbalance. The first two results we prove are adapted from the proof of Lemma 3.1.1 from Lokshtanov et al. [104], who proved that the imbalance of any graph is at least twice its cutwidth. Following these results, we show how to re-order two sets of true twins in a cutwidth-minimal ordering using an imbalance argument.

First, we show that we can consider an arbitrary ordering, rather than optimal orderings, to show that any *ordering* has imbalance at least twice its cutwidth.

Lemma 3.4.1. *If is an ordering of a graph, then* $\frac{im(\ )}{2}$ $cw(\ )$.

Proof. This proof is an adaptation of the proof of Lemma 3.1.1. If j is an index at which $c\ (\ (j))$ is as large as possible with respect to , then $n\ \ j$ is an index at which $c_R(\ (n\ \ j))$ is the maximum with respect to the reverse ordering ^{R}. By Observation 3.2.7,

$$cw(\ ) = c\ (\ (j)) = \sum_{i=1}^{\cdot} \text{rank}\ (\ (i));$$

and

$$cw(\ ^R) = c_R(\ ^R(n\ \ j)) = \sum_{i=1}^{n-j} \text{rank}_R(\ ^R(i)):$$

Now observe that

$$\begin{aligned} im(\ ) &= \sum_{i=1}^{} (\ (i)) + \sum_{i=j+1}^{} (\ (i)) \\ &= \sum_{i=1}^{} (\ (i)) + \sum_{i=1}^{n-j} {}_R(\ ^R(i)) \\ &\quad \sum_{i=1}^{} \operatorname{rank}(\ (i)) + \sum_{i=1}^{n-j} \operatorname{rank}{}_R(\ ^R(i)) \\ &= cw(\ ) + cw(\ ^R) \\ &= 2cw(\ ); \end{aligned}$$

where the last equality holds by Observation 3.2.5. □

The next result enables some results from Cutwidth to immediately carry over to Imbalance, as we will see in the next section. The result can be seen to strengthen Lemma 3.1.1 in some special cases. For the proof, recall that $c\ (v)$ is the size of the cut after vertex v with respect to .

Lemma 3.4.2. *Let G be a graph with an ordering and let j be an index of a largest cut in . If rank $(\ (i)) \quad 0$ for all $1 \quad i \quad j$ and rank $(\ (i)) \quad 0$ for all $j < i \quad n$, then $im(\ ) = 2cw(\ )$. Moreover, if $cw(\ ) = cw(G)$, then $im(G) = 2cw(G)$.*

Proof. This proof is an adaptation of the proof of Lemma 3.1.1. Since j is an index at which $c\ (\ (j))$ is as large as possible with respect to , $n \quad j$ is an index at which $c\ {}_R(\ (n \quad j))$ is the maximum with respect to the reverse ordering ^{R}. By Observation 3.2.7,

$$cw(\ ) = c\ (\ (j)) = \sum_{i=1}^{} \operatorname{rank}(\ (i));$$

and

$$cw(\ ^R) = c\ {}_R(\ ^R(n \quad j)) = \sum_{i=1}^{n-j} \operatorname{rank}{}_R(\ ^R(i)).$$

Since $(v) = |\operatorname{rank}(v)|$ and rank $(\ (i)) \quad 0$ for all $1 \quad i \quad j$ by assumption, $(\ (i)) = \operatorname{rank}(\ (i))$ for all $1 \quad i \quad j$. Similarly, rank $(\ (i)) \quad 0$ implies rank ${}_R(\ (i)) =$

$(\ (i))$ for all $j < i \quad n$, which in turn implies that $\quad_R(\ ^R(i)) = \text{rank}\ _R(\ ^R(i))$ for $1 \quad i \quad n \quad j$.

Putting it all together,

$$\begin{aligned} im(\ ) &= \sum_{i=1} (\ (i)) + \sum_{i=j+1} (\ (i)) \\ &= \sum_{i=1} (\ (i)) + \sum_{i=1}^{n-j} {}_R(\ ^R(i)) \\ &= \sum_{i=1} \text{rank}\ (\ (i)) + \sum_{i=1}^{n-j} \text{rank}\ _R(\ ^R(i)) \\ &= cw(\ ) + cw(\ ^R) \\ &= 2cw(\ ); \end{aligned} \tag{3.1}$$

where the last equality holds by Observation 3.2.5. This proves the first claim of the Lemma.

Now suppose additionally that $cw(\ ) = cw(G)$. For any ordering $\ , im(G) \quad im(\ )$. By Lemma 3.1.1, $2cw(\ ) = 2cw(G) \quad im(G)$. Therefore,

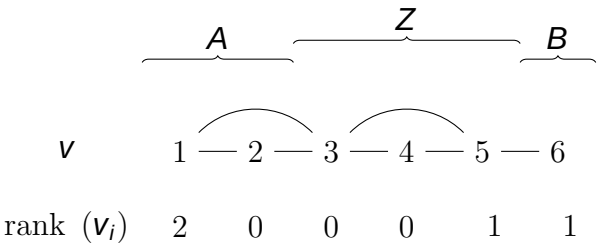
$$im(G) \underset{\text{def.}}{} im(\ ) \underset{\text{by 3.1}}{=} 2cw(\ ) = 2cw(G) \underset{\text{Lemma 3.1.1}}{} im(G);$$

and we must have equality. Namely, $im(G) = 2cw(G)$. □

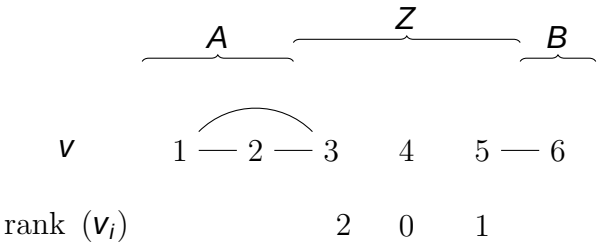
The remainder of the section shows that we can always "untangle" two sets of true twins in either an optimal ordering for imbalance or cutwidth. We start by proving this is the case for imbalance, and then show that this can be used to prove the corresponding result for cutwidth. For this result, the proof idea for an imbalance result leads to a proof for a cutwidth result.

Let Z be a clique in a graph $G = (V; E)$ and let $\quad = hA \quad Z \quad Bi$ for some sets A and B. Let $G^{\emptyset} = (V; E^{\emptyset})$ where $E^{\emptyset} = E \cap (Z \quad Z)$, that is, $G^{\emptyset}$ is the graph where all edges which have both endpoints in Z have been removed. By construction, Z is an independent set in $G^{\emptyset}$, $N_G(z) \setminus A = N_{G^{\emptyset}}(z) \setminus A$, and $N_G(z) \setminus B = N_{G^{\emptyset}}(z) \setminus B$. See Figure 3.3 for an illustration.

Recall that $\ _X$ denotes the ordering of the set X imposed by $\ $.



(a) An ordering of a graph G, where $= hA \;\; Z \;\; Bi$ and Z is a clique.



(b) An illustration of the graph $G^{\emptyset}$. The ranks of the vertices in $A \; [\; B$ are not relevant.

Figure 3.3: Illustrating the preference function for rearranging vertices in a clique without increasing the imbalance of the ordering.

Lemma 3.4.3. *Let $G = (V; E)$ be a graph with a clique Z and let $G' = (V; E')$ where $E' = E \cap (Z \times Z)$ (as above). Let σ be an ordering of G where $\sigma = \langle A, Z, B \rangle$ (for some sets $A; B \subseteq V$), and let τ be an ordering of G' which agrees with σ. Suppose that $x < y$ are consecutive in τ for some vertices $x; y \in Z$. If rank $(x) \leq$ rank (y) and σ is obtained from τ by swapping the positions of x and y, then $\mathrm{im}(\tau) \leq \mathrm{im}(\sigma)$.*

Proof. We will show that $\mathrm{im}(\tau, \{x; y\}) \leq \mathrm{im}(\sigma, \{x; y\})$, which implies that $\mathrm{im}(\tau) \leq \mathrm{im}(\sigma)$ as the imbalances of A, B and $Z \setminus \{x; y\}$ are the same in τ and σ.

For the ordering τ let z be the q'th vertex of Z $(1 \leq q' \leq |Z|)$. Observe that succ $(z) = |N_G(z) \setminus B| + (|Z| - q')$: the successors of z are all of its neighbours in B, along with every vertex in the clique Z after z, of which there are $|Z| - q'$. Similarly, pred $(z) = |N_G(z) \setminus A| + (q' - 1)$: the predecessors of z are all of its neighbours in A, along with every vertex in the clique Z before z, of which there are $q' - 1$.

Suppose that $x \in Z$ and $y \in Z$ occur at positions q and $q + 1$ of Z (i.e., $1 \leq q < |Z|$). There are two cases based on whether the inequality of rank $(x) \leq$ rank (y) is strict.

Case 1: rank $(x) =$ rank (y). We show that swapping x and y does not change the imbalance of the ordering. Consider the ranks of x and y:

$$\begin{aligned}
\text{rank } (x) &= \text{succ } (x) - \text{pred } (x) \\
&= (|N_G(x) \setminus B| + (|Z| - q)) - (|N_G(x) \setminus A| + (q - 1)) \\
&= |N_G(x) \setminus B| + (|Z| - q) - |N_G(x) \setminus A| - (q - 1) \\
&= |N_G(x) \setminus B| - |N_G(x) \setminus A| + (|Z| - q) - (q - 1) \\
&= |N_{G'}(x) \setminus B| - |N_{G'}(x) \setminus A| + (|Z| - q) - (q - 1) \\
&= \text{rank } (x) + (|Z| - q) - (q - 1) \\
&= \text{rank } (y) + (|Z| - q) - (q - 1) \text{ by case assumption} \\
&= |N_{G'}(y) \setminus B| - |N_{G'}(y) \setminus A| + (|Z| - q) - (q - 1) \\
&= |N_G(y) \setminus B| - |N_G(y) \setminus A| + (|Z| - q) - (q - 1) \\
&= |N_G(y) \setminus B| + (|Z| - q) - |N_G(y) \setminus A| - (q - 1) \\
&= (|N_G(y) \setminus B| + (|Z| - q)) - (|N_G(y) \setminus A| + (q - 1) \\
&= \text{succ } (y) - \text{pred } (y) \\
&= \text{rank } (y);
\end{aligned}$$

and

$$\begin{aligned}
\text{rank}\ (y) &= \text{succ}\ (y)\quad \text{pred}\ (y)\\
&= (jN_G(y) \setminus Bj + (jZj\quad (q+1)))\quad (jN_G(y) \setminus Aj + ((q+1)\quad 1))\\
&= (jN_G(y) \setminus Bj + (jZj\quad (q+1)))\quad (jN_G(y) \setminus Aj + q)\\
&= jN_G(y) \setminus Bj + (jZj\quad (q+1))\quad jN_G(y) \setminus Aj\quad q)\\
&= jN_G(y) \setminus Bj\quad jN_G(y) \setminus Aj + (jZj\quad (q+1))\quad q)\\
&= jN_{G^0}(y) \setminus Bj\quad jN_{G^0}(y) \setminus Aj + (jZj\quad (q+1))\quad q\\
&= \text{rank}\ (y) + (jZj\quad (q+1))\quad q\\
&= \text{rank}\ (x) + (jZj\quad (q+1))\quad q \text{ by case assumption}\\
&= jN_{G^0}(x) \setminus Bj\quad jN_{G^0}(x) \setminus Aj + (jZj\quad (q+1))\quad q\\
&= jN_G(x) \setminus Bj\quad jN_G(x) \setminus Aj + (jZj\quad (q+1))\quad q\\
&= jN_G(x) \setminus Bj + (jZj\quad (q+1))\quad jN_G(x) \setminus Aj\quad q\\
&= (jN_G(x) \setminus Bj + (jZj\quad (q+1)))\quad (jN_G(x) \setminus Aj + q)\\
&= (jN_G(x) \setminus Bj + (jZj\quad (q+1)))\quad (jN_G(x) \setminus Aj + ((q+1)\quad 1))\\
&= \text{succ}\ (x)\quad \text{pred}\ (x)\\
&= \text{rank}\ (x)\textit{:}
\end{aligned}$$

Therefore, $(Z) = \quad (Z)$ if x and y are swapped.

Case 2: rank $(x) >$ rank (y). We show that exchanging x and y does not increase the imbalance in the resulting ordering . By Lemma 3.2.2, rank $(x) =$ rank $(x)\quad 2$ and rank $(y) =$ rank $(y) + 2$.

Note that

$$\begin{aligned}
(jZj\quad (q+1))\quad ((q+1)\quad 1) &= (jZj\quad q\quad 1)\quad (q+1\quad 1)\\
&= jZj\quad q\quad 1\quad q\quad 1+1\\
&= (jZj\quad q)\quad q+1\quad 2\\
&= (jZj\quad q)\quad (q\quad 1)\quad 2\textit{:} \qquad (3.2)
\end{aligned}$$

Using the case assumption and the equation above, we can now establish the following

relationship between rank (x) and rank (y):

$$\begin{aligned}
\text{rank}\ (x) &= \text{succ}\ (x) \quad \text{pred}\ (x) \\
&= (jN_G(x) \setminus Bj + (jZj \quad q)) \quad (jN_G(x) \setminus Aj + (q \quad 1)) \\
&= jN_G(x) \setminus Bj + (jZj \quad q) \quad jN_G(x) \setminus Aj \quad (q \quad 1) \\
&= jN_G(x) \setminus Bj \quad jN_G(x) \setminus Aj + (jZj \quad q) \quad (q \quad 1) \\
&= jN_{G^0}(x) \setminus Bj \quad jN_{G^0}(x) \setminus Aj + (jZj \quad q) \quad (q \quad 1) \\
&= \text{rank}\ (x) + (jZj \quad q) \quad (q \quad 1) \\
&= \text{rank}\ (x) + (jZj \quad (q+1)) \quad ((q+1) \quad 1) + 2 \text{ by (3.2)} \\
&> \text{rank}\ (y) + (jZj \quad (q+1)) \quad ((q+1) \quad 1) + 2 \text{ by case assumption} \\
&= jN_{G^0}(y) \setminus Bj \quad jN_{G^0}(y) \setminus Aj + (jZj \quad (q+1)) \quad ((q+1) \quad 1) + 2 \\
&= jN_G(y) \setminus Bj \quad jN_G(y) \setminus Aj + (jZj \quad (q+1)) \quad ((q+1) \quad 1) + 2 \\
&= jN_G(y) \setminus Bj + (jZj \quad (q+1)) \quad jN_G(y) \setminus Aj \quad ((q+1) \quad 1) + 2 \\
&= (jN_G(y) \setminus Bj + (jZj \quad (q+1)) \quad (jN_G(y) \setminus Aj + ((q+1) \quad 1) + 2 \\
&= \text{succ}\ (y) \quad \text{pred}\ (y) + 2 \\
&= \text{rank}\ (y) + 2;
\end{aligned}$$

which is to say that rank $(x) >$ rank $(y) + 2$.

We now show that $(Z) \quad (Z)$, based on the value of rank (x); $(z) = (z)$ for all $z \in Z \setminus \{x; y\}$.

- If rank $(x) > 1$, then since rank $(x) =$ rank $(x) \quad 2 \quad 0$, $(x) = (x) \quad 2$. Since the imbalance of y can get worse by at most two, $(Z) \quad (Z)$.

- If rank $(x) = 1$, then

$$1 = \text{rank}\ (x) > \text{rank}\ (y) + 2 \ =) \quad 1 > \text{rank}\ (y):$$

  Combined with the fact that rank $(y) =$ rank $(y) + 2$, $(y) = {}_z(y) \quad 2$. Since the imbalance of x will stay the same, $(Z) < (Z)$.

- If rank $(x) < 1$, then

$$1 > \text{rank}\ (x) > \text{rank}\ (y) + 2 \ =) \quad 1 > \text{rank}\ (y):$$

  Combined with the fact that rank $(y) =$ rank $(y) + 2$, $(y) = (y) \quad 2$. Since the imbalance of x will get worse by two, $(Z) = (Z)$.

Therefore, (Z) (Z) in the case rank (x) > rank (y).

We have shown that (Z) (Z) in every case, and therefore $im(\ )$ $im(\ )$. □

Theorem 3.4.4. *Let $G = (V; E)$ be a graph. Let $Z = X [Y$ where X V and Y V are sets of true twins. If Z occurs consecutively in an ordering of G, then there is an ordering of the same (or smaller) imbalance in which X appears consecutively and Y appears consecutively (and only the vertices of Z may have been rearranged).*

Proof. Since X and Y are sets of true twins, any vertex y 2 Y is either adjacent to all vertices of X or no vertices of X. Thus we have two cases: either $X \setminus N(Y) = ;$, or X $N(Y)$.

Case 1: $X \setminus N(Y) = ;$. Since any x 2 X is not adjacent to any y 2 Y, we can swap any x past any y without changing the imbalance of the ordering. In particular, if $fx; yg$ is consecutive and $y <$ x, then swapping the positions of x and y does not change the rank of either vertex since $(x; y)$ 2 E (Observation 3.2.1). Therefore, we can put all the vertices of X before all vertices of Y in Z, and we are done.

Case 2: X $N(Y)$. In this case, Z is a clique as each set X and Y is a clique, and every vertex in Y is adjacent to every vertex in X. Let $G^0 = (V; E^0)$ where $E^0 = E$ n $(Z$ $Z)$, and let be an ordering of G^0 which agrees with . The function rank () takes on at most two values as vertices in Z are in at least one of X or Y. By Lemma 3.4.3, the imbalance is minimized by putting the twins with the lower value of rank () to the left. If rank () takes only one value, then any ordering of Z has the same total imbalance. □

We now show how this can be used to untangle two sets of true twins in a cutwidth ordering. The proof is illustrated in Figure 3.4.

Lemma 3.4.5. *Let X and Y be (not necessarily maximal) sets of true twins in an ordering , where Y is partitioned into two sets Y_1 and Y_2. If $= hL$ Y_1 X Y_2 Ri for some sets L and R, then there is an ordering ^{0} where $^0 = hL$ X Y Ri or $^0 = hL$ Y X Ri and $cw(\ ^0)$ $cw(\ )$.*

Proof. First, we may assume that Y $N(X)$, as otherwise moving any vertex of Y_2 backward to before X does not change its rank, and we may freely do so to get a desired ordering.

Let $` = jLj$.

Claim 3.4.6. *Either*

1. *there is an ordering* $= hL\ X\ [\ Y\ Ri$ *such that* $cw(\ ) \quad cw(\ )$ *and there is index* $` \quad j \quad ` + jXj + jYj$ *such that for* $` \quad i \quad j$, *rank* $(v_i) \quad 0$ *and for* $j < i \quad ` + jXj + jYj$, *rank* $(v_i) \quad 0$, *or,*

2. *we can find an ordering which is desired by the lemma.*

Proof of claim: Let $x_1 \ 2\ X$ be the leftmost vertex of X in $\ $, and let $x_2 \ 2\ X$ be the rightmost vertex of X in $\ $. We proceed with cases based on the signs of rank (x_1) and rank (x_2).

Case 1: rank $(x_2) \quad 0$. By Observation 3.2.10, rank $(x_1) > 0$. By Lemma 3.2.11, we can move X forward to its rightmost position where x_2 achieves a non-negative rank, to get an ordering $\ = hL\ Y_1\ Y_2^0\ X\ Y_2^{00}\ Ri$ where Y_2^0 and Y_2^{00} partition Y_2. If Y_2^{00} is empty, then $\ $ is an ordering desired by the lemma. So suppose that Y_2^{00} is non-empty. Let $Y_2^0 = hy_1^0; : : : ; y_{jY_2^0j}^0 i$ and $Y_2^{00} = hy_1^{00}; : : : ; y_{jY_2^{00}j}^{00} i$.

Case 1a: rank $(y_{jY_2^0j}^0) \quad 0$. By Observation 3.2.10, $0 \quad$ rank $(y_{jY_2^0j}^0) \quad$ rank (y_1^{00}). In this case, all vertices of Y_2^{00} have non-positive rank in $\ $, and by Lemma 3.2.11, all of Y_2^{00} can be moved backward to be immediately right of Y_2^0, in which case the resulting order is one desired by the lemma (it has the form $hL\ \ Y_1\ \ Y_2^0\ \ Y_2^{00}\ \ X\ \ Ri$).

Case 1b: rank $(y_{jY_2^0j}^0) > 0$. If rank $(y_1^{00}) \quad 0$, then by Observation 3.2.10, all vertices of $Y_1\ [\ Y_2^0$ also have non-negative rank, and by Lemma 3.2.11, we can move $Y_1\ [\ Y_2^0$ to be immediately left of Y_2. The resulting ordering is one desired by the lemma (it has the form $hL\ \ X\ \ Y_1\ \ Y_2^0\ \ Y_2^{00}\ \ Ri$). Otherwise, rank $(y_1^{00}) < 0$ and we can take $j = ` + jY_1j + jY_2^0j + jXj$ along with $\ $. Since the moving the vertices did not increase the cutwidth (by Lemma 3.2.11), $cw(\ ) \quad cw(\ )$.

Case 2: rank $(x_2) < 0$ and rank $(x_1) \quad 0$. By Lemma 3.2.11, we can move X backward to its leftmost position where x_1 achieves a non-positive rank, to get an ordering $\ = hL\ \ Y_1^0\ \ X\ \ Y_1^{00}\ \ Y_2\ \ Ri$ where Y_1^0 and Y_1^{00} partition Y_1. If Y_1^0 is empty, then $\ $ is an ordering desired by the lemma. So suppose that Y_1^0 is non-empty. Let $Y_1^0 = hy_1^0; : : : ; y_{jY_1^0j}^0 i$ and $Y_1^{00} = hy_1^{00}; : : : ; y_{jY_1^{00}j}^{00} i$.

Case 2a: rank $(y_1^{00}) \quad 0$. In this case, all vertices of Y_1^0 have non-negative rank in $\ $, and by Lemma 3.2.11, all of Y_1^0 can be moved forward to be immediately left of Y_1^{00}, in which case the resulting order is one desired by the lemma (it has the form $hL\ \ X\ \ Y_1^0\ \ Y_1^{00}\ \ Y_2\ \ Ri$).

Case 2b: rank $(y_1^{00}) < 0$. If rank $(y_{jY_1^0j}^0) \quad 0$, then by Observation 3.2.10 all vertices of $Y_1^{00}\ [\ Y_2$ have non-positive rank, and by Lemma 3.2.11, $Y_1^{00}\ [\ Y_2$ can be moved backward to be immediately right of Y_1^0. The resulting ordering is one desired by the lemma (it has the

form $hL\ Y_1'\ Y_1''\ Y_2\ X\ Ri$). Otherwise, rank $(y'_{jY_1'j}) > 0$ and we can take $j = `+jY_1'j$ along with . Since the moving the vertices did not increase the cutwidth (by Lemma 3.2.11), $cw(\ ) \quad cw(\ )$.

Case 3: rank $(x_2) < 0$ and rank $(x_1) > 0$. By Observation 3.2.10, the ranks of X are non-increasing in . Let $y_1 \in Y_1$ be the rightmost vertex of Y_1 in , and let $y_2 \in Y_2$ be the leftmost vertex of Y_2 in .

If rank $(y_1) \quad 0$, then by Observation 3.2.10, all vertices of Y_2 also have non-positive rank, and we can move Y_2 to be immediately right of Y_1 by Lemma 3.2.11. Similarly, if rank $(y_2) \quad 0$, then by Observation 3.2.10, all vertices of Y_1 also have non-negative rank, and we can move Y_1 to be immediately left of Y_2 by Lemma 3.2.11. Therefore, we may assume that the sign of rank (y_1) and rank (y_2) are different, and by Observation 3.2.10 the only option is that rank $(y_1) \quad 0 \quad$ rank (y_2). Thus, there must be an index $` + jY_1j < j < ` + jY_1j + jXj$ that satisfies the requirement, since the signs of x_1 and x_2 are different. Thus we can take $=$ along with j (and obviously $cw(\ ) = cw(\ ) \quad cw(\ )$), proving the claim.

Now let be an ordering provided by Claim 3.4.6.

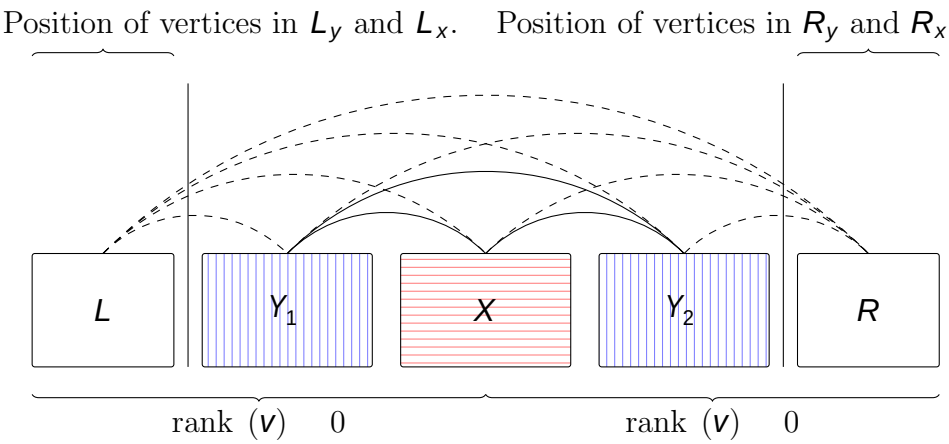
Let $L_y = N(y) \setminus L$ and $R_y = N(y) \setminus R$ for any $y \in Y$. Let $L_x = N(x) \setminus L$ and $R_x = N(x) \setminus R$ for any $x \in X$. Let O be the edges passing over $X \cup Y$ in : namely, edges $(u; v)$ such that $u \in L$ and $v \in R$. We can count the edges over a cut between two consecutive vertices $x; y$ of $X \cup Y$ using these sets.

We define a subgraph $G' = (V'; E')$ of $G = (V; E)$ by taking $V' = V$ and $E' = E \cap (E_L \cup E_R)$ where $E_L = \{(u; v) \in E \mid u; v \in L\}$ and $E_R = \{(u; v) \in E \mid u; v \in R\}$. Let be the ordering of G' that agrees with ; we can again write $= hL\ Y_1\ X\ Y_2\ Ri$ for some sets Y_1 and Y_2 that partition Y (see Figure 3.4).

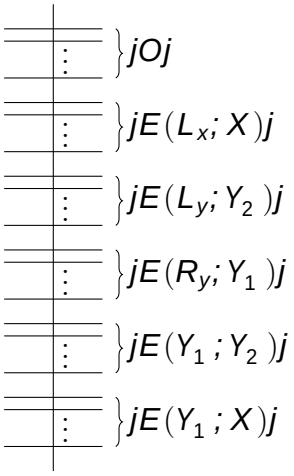
Claim 3.4.7. *Either,*

- *the maximum cut of is after $c(v_i)$ for some $` \quad i \quad ` + jXj + jY j$ and there is an index j of such that all ranks are non-negative before j and all ranks are non-positive after j, or,*
- *we can nd an ordering which is desired by the lemma.*

Proof of claim: First, observe that since there are no edges within L, for $1 \quad i < `$, $c(v_i) \quad c(v_{i+1})$, as $c(v_{i+1})$ has at least as many edges as $c(v_i)$ but may also have those ends which end at $v_{i+1}$. Second, since there are no edges within $R$, for $` + jY j + jXj \quad i < j \quad j$,



(a) Initial ordering of G^0 provided by as in the proof of Lemma 3.4.5. Note that it is not necessarily the case that $L_x = L_y$ or that $R_y = R_x$, however, both L_y and L_x are to the left of Y_1 and both R_x and R_y are to the right of Y_2. The boxes of $Y = Y_1 \ [\ Y_2$ (vertical lines, blue) are all vertices of Y are true twins, and X (horizontal lines, red), is another set of true twins. The set L and R are independent sets. We will show that we can rearrange the vertices between the two vertical lines without increasing the cutwidth of the ordering. Dashed lines indicate some edges are present between vertices in one box and vertices in the other box.



(b) A close look at the cut c (y) where y is the rightmost vertex of Y_1 in . Each horizontal line is part of an edge drawn over the cut, indicated by the vertical line. The edges crossing the cut are grouped based on where their endpoints are in ; recall that $E(A; B)$ indicates edges with one endpoint in A and the other in B.

Figure 3.4: An illustration of G^0 provided by as in the proof of Lemma 3.4.5. (a) shows the overall structure of the ordering, grouped into sets. (b) shows an example cut between two vertices in $X \ [\ Y$.

$c(v_i)$ $c(v_{i+1})$, as $c(v_i)$ has at least as many edges as $c(v_i)$ but may also have those ends which end at v_i. Therefore,

$$\max_{1 \; i \; j \; j} fc(v_i)g = \; _{\backprime} \; _{i} \max_{\backprime + jYj + jXj} fc(v_i)g;$$

as required.

We now prove the second part of the claim. Note that for $v \in L$, rank (v) 0: if v is not isolated, its entire neighbourhood is in $R \cup X \cup Y$ and $v <$ $(R \cup X \cup Y)$. Similarly, for $v \in R$, rank (v) 0: if v is not isolated, its entire neighbourhood is in $L \cup X \cup Y$ and $(L \cup X \cup Y) < v$.

For all $v \in X \cup Y$, rank $(v) =$ rank (v) as the edges from v to its neighbours have not been removed in $G^{\emptyset}$. By Claim 3.4.6, the required j exists (or the desired ordering exists).

Consider the cut $c(v)$ for any $v \in V(G) = V(G^{\emptyset})$: $c(v) = c(v)$ for some number , which represents the difference in the number of edges crossing $c(v)$ but not $c(v)$. Since $G^{\emptyset}$ did not add any new edges, 0. Therefore, $c(v)$ $c(v)$ for all $v \in V$, and in particular by Claim 3.4.6,

$$cw(\;) \quad cw(\;) \quad cw(\;): \tag{3.3}$$

Since X and Y are true twins, by Theorem 3.4.4, there is another ordering of $G^{\emptyset}$ such that $X <$ Y or $Y <$ X, and $im(\;)$ $im(\;)$. Furthermore, only vertices in $X \cup Y$ have been rearranged.

Claim 3.4.8. $cw(\;)$ $cw(\;)$.

Proof of claim: By Claim 3.4.7 and Lemma 3.4.2, $im(\;) = 2cw(\;)$. By Lemma 3.4.1,

$$cw(\;) \underset{\text{Lemma 3.4.1}}{\frac{im(\;)}{2}} \underset{\text{Theorem 3.4.4}}{\frac{im(\;)}{2}} \underset{\text{Claim 3.4.7 and Lemma 3.4.2}}{=} cw(\;):$$

By (3.3), $cw(\;)$ $cw(\;)$ $cw(\;)$ $cw(\;)$, as required.

Since X and Y are each consecutive in and $cw(\;)$ $cw(\;)$, we can replace $_{X \cup Y}$ with $_{X \cup Y}$ without increasing the cutwidth of the ordering; the resulting ordering proves the lemma. □

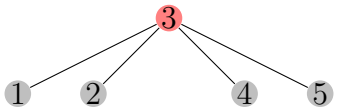


Figure 3.5: A graph that shows Theorem 3.5.1 is not applicable for false twins. Any complete bipartite graph $K_{1;m}$ with m 2 vertices must split the leaves into two consecutive sets in any cutwidth-minimal ordering. In particular, $bm{=}2c$ vertices of the larger partite set must come before the single vertex in the smaller partite set. In this case, $= h1; 2; 3; 4; 5i$ is such that $cw(G) = cw( ) = 2$.

3.5 A Structure of Optimal Orderings

In this section, we show that for any graph G, there is a cutwidth-minimal ordering of G in which, for each equivalence class of true twins S, the vertices of S appear consecutively. This is helpful for FPT algorithms (e.g., Section 3.6), and may be of general interest. The theorem is not applicable for false twins; Figure 3.5 shows a graph for which any cutwidth-minimal ordering must split a set of false twins.

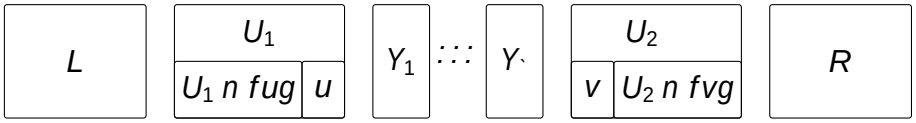
Theorem 3.5.1. *For any graph G, there exists a cutwidth-minimal ordering of $V(G)$ such that each equivalence class of true twins appears consecutively in .*

Define ( ) to be the number of maximal consecutive sets of true twins in the ordering . Define $tt(G)$ to be the number of equivalence classes of true twins in the graph G. By definition, ( ) $tt(G)$ for any graph G and ordering of G.

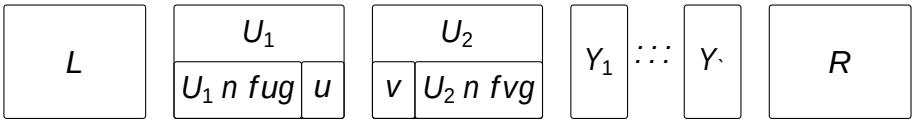
Proof of Theorem 3.5.1. Say that | if and only if ( ) < ( ) and $cw( )$ $cw( )$ for two orderings and of G.

Suppose that is such that $cw( ) = cw(G) = k$ but ( ) > $tt(G)$ (if ( ) = $tt(G)$ and $cw( ) = cw(G)$ there is nothing to prove). We will find an ordering | . Repeating this until the resulting order is such that ( ) = $tt(G)$ proves the theorem.

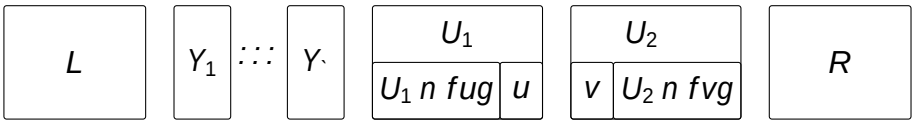
Let U $V(G)$ be an equivalence class of true twins which is not consecutive in . We may assume that there are at least two distinct, maximal consecutive sets of vertices U_1 U and U_2 U in . Pick two such sets U_1 and U_2 such that $U_1 <$ U_2 and there is no vertex q 2 U such that $U_1 <$ $q <$ U_2. Let u 2 U_1 be the rightmost vertex of U_1 in , and v 2 U_2 be the leftmost vertex of U_2 in . We have that $u <$ v, and that u is not beside v in (as otherwise U_1 would be beside U_2 and U_1 [U_2 would be a consecutive set in).



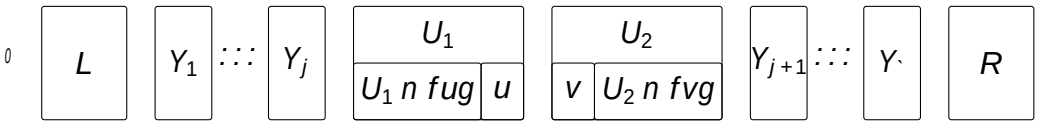
(a) Initial ordering of as in the proof of Theorem 3.5.1.



(b) The resulting ordering when rank (u) 0 and rank (v) 0. In this case, all the vertices of U_2 must have had non-positive rank in by Observation 3.2.10, and all could be moved backwards by Lemma 3.2.12



(c) The resulting ordering when rank (u) > 0 and rank (v) 0. In this case, all the vertices of U_1 must have had non-negative rank in by Observation 3.2.10, and all could be moved forwards by Lemma 3.2.12



(d) One possible resulting ordering when rank (u) > 0 and rank (v) < 0. The actual order depends on what is provided by Lemma 3.4.5: in particular, the positions of Y_j and U_1 [U_2 may be swapped.

Figure 3.6: An illustration of the proof of Theorem 3.5.1.

Write $= \langle L\ U_1\ Y\ U_2\ R \rangle$ for some set Y and (possibly empty) sets L and R. Furthermore, partition Y into maximal consecutive sets $Y_1, \ldots, Y_\ell$ where each Y_j ($1 \leq j \leq \ell$) is a set of true twins from the same equivalence class. Since U_1 is not beside U_2 in , $\ell \geq 1$. This is illustrated in Figure 3.6a.

We consider cases based on the signs of rank (u) and rank (v). Since u and v are true twins, by Observation 3.2.10, the case where rank $(u) \geq 0$ but rank $(v) < 0$ is impossible.

Let $U_1 = \{u_i, \ldots, u_j = u\}$ and $U_2 = \{v = u_{j+1}, \ldots, u_k\}$ for some positive integers i, j, and k. Then we can see that $= \langle L\ \{u_i, \ldots, u_j = u\}\ Y\ \{v = u_{j+1}, \ldots, u_k\}\ R \rangle$.

Case 1: rank $(u) \leq 0$ and rank $(v) \leq 0$. By Lemma 3.2.12, we can move all of U_2 backwards, to be immediately right of u, without increasing the cutwidth of the ordering. That is, if is obtained by moving U_2 immediately right of u (and therefore U_1), $( ) = ( )$ 1 and $cw( ) \leq cw( )$ and $\in$. See Figure 3.6b. We have an ordering $\in$ in the case rank $(u) \leq 0$ and rank $(v) \leq 0$.

Case 2: rank $(u) > 0$ and rank $(v) \leq 0$. By Lemma 3.2.9, rank $(u) \geq$ rank $(v) + 2$. Therefore, moving u forward from its position in to be immediately left of v results in a non-negative rank > 1. The resulting ordering has the form $\langle L\ \{u_i, \ldots, u_{j-1}\}\ Y\ \{u_j = u, v = u_{j+1}, \ldots, u_k\}\ R \rangle$. By Lemma 3.2.12, we can move all of U_1 forwards, to be immediately left of v, without increasing the cutwidth of the ordering. The resulting ordering has $( ) = ( )$ 1 and $cw( ) \leq cw( )$, and therefore $\in$. See Figure 3.6c. Thus we have an ordering $\in$ in the case rank $(u) > 0$ and rank $(v) \leq 0$.

Case 3: rank $(u) > 0$ and rank $(v) < 0$. There is a rightmost position between u and v to the right of u such that u will have its minimal non-negative rank. We can move u forward to that position by Lemma 3.2.11; the resulting ordering has the form $\langle L\ \{u_i, \ldots, u_{j-1}\}\ Y'\ \{u_j = u\}\ Y''\ \{v = u_{j+1}, \ldots, u_k\}\ R \rangle$ where $Y = Y' \cup Y''$. We can then move all of U_1 forward by Lemma 3.2.12. Call the resulting ordering $'$. Then, the position immediately right of u in $'$ is where v achieves its maximal negative rank. We can move v backward to that position by Lemma 3.2.11; the resulting ordering has the form $\langle L\ Y'\ \{u_i, \ldots, u_{j-1}, u_j = u\}\ \{u_{j+1} = v\}\ Y''\ \{u_{j+2}, \ldots, u_k\}\ R \rangle$. We can then move all of U_2 backward by Lemma 3.2.12. Call the resulting ordering $''$.

We have reduced by 1 by gathering the sets U_1 and U_2, but this may have resulted in splitting some set Y_j ($1 \leq j \leq \ell$), which may have increased by 1. We may assume therefore that $('') = ( )$, as otherwise $('') < ( )$ and $cw('') \leq cw( )$; therefore $'' \in$ and we are done.

Therefore, we must have that $'' = \langle L' Y_j' \ U_1\ U_2\ Y_j''\ R' \rangle$ where Y_j' and Y_j'' partition Y_j, $L' = L \cup \bigcup_{i=1}^{j-1} Y_i$, and $R' = R \cup \bigcup_{i=j+1}^{\ell} Y_i$. By Lemma 3.4.5 and the fact that

$U_1 \cup U_2$ and Y_j are sets of true twins, there is an ordering where $\pi' = \pi L^{\pi} Y_j U_1 \cup U_2 R^{\pi} i$ or $\pi' = \pi L^{\pi} U_1 \cup U_2 Y_j R^{\pi} i$ and $cw(\pi') \leq cw(\pi^1)$. Thus, $cw(\pi') = cw(\pi^1) \leq 1$ and $cw(\pi') \leq cw(\pi^1) \leq cw(\sigma)$ and $\pi' \in \Pi$, as required. See Figure 3.6d for one possible visualization. Thus we have an ordering $\pi' \in \Pi$ in the case rank $(u) > 0$ and rank $(v) < 0$.

Since we have found an ordering $\sigma \in \Pi$ in every case, the proof is complete. □

3.6 Edge Clique Cover Number Parameterization

In this section, we show that Theorem 3.5.1 immediately yields an FPT algorithm when the parameter is the edge clique cover number of the graph.

Theorem 3.6.1. Cutwidth *is fixed-parameter tractable when parameterized by the edge clique cover number of the graph.*

Proof. Let $k = cc(G)$. By Theorem 3.5.1, there exists a cutwidth-minimal ordering in which each equivalence class of true twins appears consecutively. The ordering of vertices within each equivalence class of true twins can be arbitrary. Gramm et al. [71] proved that graphs with edge clique cover at most k have at most 2^k different equivalence classes of true twins (cf. Theorem 2.5.6). Note that each vertex must be in some equivalence class of true twins (which may be a singleton set). Thus, an algorithm can try each of the $O(2^k!)$ orderings and output the ordering with the smallest cutwidth. □

3.7 Restricted Twin Cover Number Parameterization

In this section, we show that there is an FPT algorithm for Cutwidth when the parameter is the restricted twin cover number. We use the FPT algorithm of Fellows et al. [52] for graphs with bounded vertex cover number as inspiration, and Theorem 3.5.1.

We formulate Cutwidth as an instance of the following problem, which will also be called p-Opt-ILP for short. The input to Cutwidth parameterized by the vertex cover number of the graph will be a graph G and a vertex cover C of G such that $|C| \leq k$. If C is not provided but $vc(G) \leq k$, then C can be found in time $O(2^k n)$ (see, e.g., Theorem 3.2.1, Downey and Fellows [46]).

Problem 3.7.1 (p-Variable Integer Linear Programming Optimization)**.** *Let matrices $A \in \mathbb{Z}^{m \times p}$, $b \in \mathbb{Z}^{m \times 1}$, and $c \in \mathbb{Z}^{1 \times p}$ be given. We want to find a vector $\bar{x} \in \mathbb{Z}^{p \times 1}$*

that minimizes the objective function $c\ \bar{x}$ and satis es the m inequalities, that is, $A\ \bar{x}\quad b$. The number of variables p is the parameter.

An instance of p-Opt-ILP is called an *integer linear program* (ILP). Fellows et al. [52] showed that p-Opt-ILP is FPT when the parameter is p.

Theorem 3.7.2 (Theorem 2, Fellows et al. [52]). *p-Opt-ILP can be solved using $O(p^{2.5p+o(p)}\ L\ \log(MN))$ arithmetic operations and space polynomial in L. Here, L is the number of bits in the input, N is the maximum of the absolute values any variable can take, and M is an upper bound on the absolute value of the minimum taken by the objective function.*

We modify the following ILP formulation for Cutwidth on graphs of bounded vertex cover number by Fellows et al. [52]. The notation is updated to match the notation used in this work, where possible; the following are notation is still required. Given an ordering $\quad$ and a vertex cover C such that $\quad_C = \langle c_1, \ldots, c_k \rangle$, a vertex $x \in V \quad C$ is at *location* i for $1 \quad i \quad k$ if i is the largest integer such that $c_i < \quad x$, or location 0 if $x < \quad c_1$. The set I will be the independent set obtained by deleting the vertex cover C, that is, $I = G \quad C$. The set L_i will denote the vertices of I at location i. Define $C_i = \{c_1, \ldots, c_i\}$ for $1 \quad i \quad k$. For a subset $S \quad C$, we define $I_S \quad G \quad C$ to be the vertices with neighbourhood S, i.e., $I_S = \{v \in G \quad C \mid N(v) = S\}$. Recall that $\quad_i$ denotes $\quad_{<(i)}$, which is the set of vertices before the ith vertex in $\quad$, and that analogous definitions are used for $>$, $\quad$, and $\quad$. Finally, define a new variant of the *rank* function as follows: $\mathrm{rank}(S, v) = |N(v) \cap S| \quad |N(v) \setminus S|$ (this is similar to the original definition of rank which we use, but computes the value relative to a set rather than an ordering).

ILP Formulation 3.7.3 (Cutwidth, Fellows et al. [52]).

> We guess[1] the ordering $c_1 < \quad < c_k$ of the vertices in [the vertex cover] C in an optimal permutation. We consider the ordering $\quad_{L_i}$ for some i between 0 and k. Suppose that $(c_i)_{\quad} = s$, then, for any t with $s < t \quad s + |L_i|$ we have that $c\ (v_t) = c\ (v_s) + \sum_{j=s+1}^{t} \mathrm{rank}(\quad_{j\ 1}, v_j)$. Since the set of vertices in the locations L_i $(0 \quad i \quad k)$ form an independent set, $\mathrm{rank}(\quad_{j\ 1}, v_j) = \mathrm{rank}(\quad_{c_i}, v_j)$ for every j between $s+1$ and t. This gives the equation $c\ (v_t) = c\ (v_s) + \sum_{j=s+1}^{t} \mathrm{rank}(\quad_{c_i}, v_j)$.
>
> Hence if we start with an optimal permutation $\quad$ and reorganize $\quad_{L_i}$ at each location i to sort the vertices by rank with respect to C_i in non-decreasing order,

[1] Note that this is done by essentially trying the following proof for every permutation of the vertex cover C, which adds a factor of $C!$ to the running time, but is allowed since $|C|$ is the parameter.

we get another optimal ordering with a fixed inner order for each location. [There may be vertices with the same rank, and for a particular rank value, the order of the vertices with that rank does not matter.] In such orderings, the largest values of [a cut] occur either at c (c_j) or c $(\ (c_j) \quad 1)$ for some j between 1 and k. Since the rank of a vertex $v \ 2 \ I$ with respect to C_i only depends on i and the neighbourhood of v, we can use this together with the fact that $c\ (v_t) = c\ (v_s) + \sum_{j=s+1}^{t} \mathrm{rank}(\ _s; v_j)$ in order to give an integer linear programming formulation for the Cutwidth problem.

For every $S \quad C$ and location i we introduce a variable x_S^i that tells us the number of vertices with neighbourhood S that are at location i. For every i between 1 and k we add a variable y_i which encodes $\mathrm{rank}(\ _{<c_i}; c_i)$ and the constant $e_i = \ jN(c_i) \setminus (C \cap C_i)j \quad jN(c_i) \setminus C_ij$. For every $S \quad C$ and location i we also compute the constant e_S^i that indicates the rank of a vertex with neighbourhood S with respect to C_i. [That is, $e_S^i = \mathrm{rank}(C_i; v)$ for any $v \ 2 \ I_S$, as all vertices of I_S are false twins.] Finally, we need a variable c that represents the cutwidth of G. For the constraints, we need to make sure the variables x_S^i represent a valid partitioning of I into $L_0; : : : ; L_k$. Finally we need constraints to encode the rank of the vertex cover vertices and the connection between the partitioning of I and the cutwidth c. [Each constraint is mapped to a line in the program after the program is formulated.] This yields the following integer linear program:

$$\min c$$

$$\text{s.t.} \quad \sum_{i=0}^{k} x_S^i = jI_Sj \qquad 8S \quad C \quad (3.4)$$

$$y_i = e_i + \sum_{S \ \ Cjc_i 2S} \ \sum_{j=i}^{k} x_S^j \ - \ \sum_{j=0}^{i-1} x_S^j \qquad 8i \ 2 \ f0; : : : ; kg \quad (3.5)$$

$$c \quad \sum_{j=0}^{k} y_j + \sum_{j=0}^{i-1} \ \sum_{S \ \ C} e_S^j \ \ x_S^j \qquad 8i \ 2 \ f1; : : : ; kg \quad (3.6)$$

$$c \quad \sum_{j=0}^{i-1} y_j + \sum_{j=0}^{i-1} \ \sum_{S \ \ C} e_S^j \ \ x_S^j \qquad 8i \ 2 \ f1; : : : ; kg \quad (3.7)$$

$$x_S^i \quad 0 \qquad 8i \ 2 \ f0; : : : ; kg; S \quad C \quad (3.8)$$

Since the value of $cw(\ )$ is bounded by n^2 and the value of any variable in the integer linear program is bounded by n^2, Theorem 3.7.2 implies that this integer linear program can be solved in FPT time.

This completes the description of the ILP for Cutwidth adapted from Fellows et al. [52]. The remainder of this section is new an additional description of the ILP constraints and the modifications needed to adapt the ILP to the restricted twin cover number.

We describe the constraints in the ILP above so that when we modify them in the proof below, it is clear what we are changing. Constraint 3.4 ensures that every vertex in the independent set $G \quad C$ is placed into some location, and Constraint 3.8 ensures that no location receives a negative number of such vertices. Constraint 3.5 represents the rank of the vertices in the vertex cover C. Finally, Constraints 3.6 and 3.7 ensure that the cutwidth is computed by counting the edges in a cut before a vertex in the cover (Constraint 3.6) and counting the edges in a cut after a vertex in the cover (Constraint 3.7).

Now we turn to Cutwidth parameterized by the restricted twin cover number. We will modify the ILP formulation above to obtain a new one. The following lemma is required for the new formulation in order to compute the maximum cut within a set of consecutive true twins.

Lemma 3.7.4. *Let Q be a set of true twins with at least two vertices which appears consecutively in some ordering $\quad$ starting at position i. If $jN(Q) \setminus \ _{>Q}j = r$, $jN(Q) \setminus \ _{<Q}j = \ `$ and $` \quad r$, then a largest cut of $c\ (i \quad 1);:::;c\ (i + (jQj \quad 1))$ occurs at cut $c\ (j)$ where $j = \max f(i \quad 1);(i \quad 1) + bjQj=2c + d(r \quad `)=2eg$.*

Proof. We can write $\quad = hL \ Q \ Ri$ for some sets of vertices L and R. Let O be the number of edges passing over all vertices Q, that is, with one endpoint in L and the other in R. Formally, $O = jE(L;R)j$.

Let $j = \max f(i \quad 1);(i \quad 1) + bjQj=2c + d(r \quad `)=2eg$. We have two cases, based on whether $bjQj=2c + d(r \quad `)=2e < 0$. Since $` \quad r$, $(r \quad `) \quad 0$.

Case 1: $bjQj=2c+d(r \quad `)=2e < 0$. We will show that $j = i \quad 1$ has the largest cut among $c\ (i \quad 1);:::;c\ (i + (jQj \quad 1))$.

We use the following facts. First, note that if jQj is even and $r \quad `$ is even,

$$\begin{aligned} & d(r \quad `)=2e > bjQj=2c \\ =) \quad & (r \quad `)=2 > jQj=2 \\ =) \quad & ` \quad r > jQj \\ =) \quad & ` > jQj + r; \end{aligned} \tag{3.9}$$

and if jQj is even and $r \quad `$ is odd,

$$\begin{aligned} & d(r \quad `)=2e > bjQj=2c \\ =) \quad & ((r \quad `)+1)=2 > jQj=2 \\ =) \quad & ((r \quad `)+1) > jQj \\ =) \quad & (r \quad `) \quad 1) > jQj \\ =) \quad & ` \quad r > jQj+1 > jQj \\ =) \quad & ` > jQj+r: \end{aligned} \tag{3.10}$$

That is, as long as jQj is even, we have from Equations 3.9 and 3.10 that

$$` > jQj+r: \tag{3.11}$$

If jQj instead is odd and $r \quad `$ is even,

$$\begin{aligned} & d(r \quad `)=2e > bjQj=2c \\ =) \quad & (r \quad `)=2 > (jQj \quad 1)=2 \\ =) \quad & ` \quad r > jQj \quad 1 \\ =) \quad & ` > jQj \quad 1+r; \end{aligned} \tag{3.12}$$

and if jQj is odd and $r \quad `$ is odd as well,

$$\begin{aligned} & d(r \quad `)=2e > bjQj=2c \\ =) \quad & ((r \quad `)+1)=2 > (jQj \quad 1)=2 \\ =) \quad & ((r \quad `)+1) > jQj \quad 1 \\ =) \quad & (r \quad `) \quad 1 > jQj \quad 1 \\ =) \quad & ` \quad r > jQj \\ =) \quad & ` > jQj+r > jQj \quad 1+r: \end{aligned} \tag{3.13}$$

That is, as long as jQj is odd, we have from Equations 3.12 and 3.13 that

$$` > jQj \quad 1+r: \tag{3.14}$$

Now, we compare the cuts between $c\ (j)$ and another cut $c\ (j+x)$, $1 \quad x \quad jQj$. If

jQj is even:

$$\begin{aligned}
c\ (j) &= `\ \ jQj + O \\
&= `(jQj + x \quad x) + O \text{ for } 1 \quad x \quad jQj \\
&= `(jQj \quad x) + `\ \ x + O \\
&> `(jQj \quad x) + (jQj + r)x + O \text{ by (3.11)} \\
&= `(jQj \quad x) + jQj \ \ x + r \ \ x + O \\
&> `(jQj \quad x) + (jQj \quad x)x + r \ \ x + O \text{ since } jQj \quad x \quad 1 \\
&= c\ (j + x):
\end{aligned}$$

If jQj is odd,

$$\begin{aligned}
c\ (j) &= `\ \ jQj + O \\
&= `(jQj + x \quad x) + O \text{ for } 1 \quad x \quad jQj \\
&= `(jQj \quad x) + `\ \ x + O \\
&> `(jQj \quad x) + (jQj \quad 1 + r)x + O \text{ by (3.14)} \\
&= `(jQj \quad x) + jQj \ \ x \quad x + r \ \ x + O \\
&= `(jQj \quad x) + (jQj \quad 1)x + r \ \ x + O \\
&\quad `(jQj \quad x) + (jQj \quad x)x + r \ \ x + O \text{ since } jQj \quad x \quad 1 \\
&= c\ (j + x):
\end{aligned}$$

Regardless of whether jQj is even or odd, the cut at $c\ (j)$ is at least as large as the cut between any two vertices of Q or after the last vertex of Q.

Case 2: $bjQj/2c + d(r \quad `)/2e \quad 0$. We will show that the cut at $j = (i \quad 1) + bjQj/2c + d(r \quad `)/2e$ is the largest in this case. Note that $(r \quad `)/2 \quad 0$ since $` \quad r$ and if we are in this case, $d(r \quad `)/2e \quad bjQj/2c$. Thus, if $jQj$ is even and $r \quad `$ is even,

$$\begin{aligned}
& d(r \quad `)/2e \quad bjQj/2c \\
=) \quad & (r \quad `)/2 \quad jQj/2 \\
=) \quad & ` \quad r \quad 1 \quad ` \quad r \quad jQj;
\end{aligned} \tag{3.15}$$

and if jQj is even but $r \quad `$ is odd,

$$\begin{aligned}
& d(r \quad `)/2e \quad bjQj/2c \\
=) \quad & ((r \quad `) + 1)/2 \quad jQj/2 \\
=) \quad & ((r \quad `) + 1) \quad jQj \\
=) \quad & (r \quad `) \quad 1 \quad jQj \\
=) \quad & ` \quad r \quad 1 \quad jQj;
\end{aligned} \tag{3.16}$$

That is, as long as jQj is even, we have from Equations 3.15 and 3.16 that

$$` \quad r \quad 1 \quad jQj: \tag{3.17}$$

If jQj is instead odd and $r \quad `$ is even,

$$\begin{aligned} & d(r \quad `)=2e \quad bjQj=2c \\ =) \quad & (r \quad `)=2 \quad (jQj \quad 1)=2 \\ =) \quad & ` \quad r \quad jQj \quad 1 \quad jQj; \end{aligned} \tag{3.18}$$

and if jQj is odd and $r \quad `$ is also odd,

$$\begin{aligned} & d(r \quad `)=2e \quad bjQj=2c \\ =) \quad & ((r \quad `)+1)=2 \quad (jQj \quad 1)=2 \\ =) \quad & ((r \quad `)+1) \quad jQj \quad 1 \\ =) \quad & (r \quad `) \quad 1 \quad jQj \quad 1 \\ =) \quad & ` \quad r \quad jQj; \end{aligned} \tag{3.19}$$

That is, as long as jQj is odd, we have from Equations 3.18 and 3.19 that

$$` \quad r \quad jQj: \tag{3.20}$$

In this case, there are $j \quad (i \quad 1) = bjQj=2c + d(r \quad `)=2e = y \quad 1$ vertices of Q to the left of $c \ (j)$. First, we handle the case that jQj is even:

$$\begin{aligned} c \ (j) &= ` \quad (jQj \quad y)+r \quad y+(y)(jQj \quad y)+O \\ &= ` \quad (jQj \quad y)+r \quad y+(y)(jQj \quad y)+(x)(jQj \quad y) \quad (x)(jQj \quad y)+O \\ &= ` \quad (jQj \quad y)+r \quad y+(x+y)(jQj \quad y) \quad (x)(jQj \quad y)+O \\ &\quad ` \quad (jQj \quad y)+r \quad y+(x+y)(jQj \quad y) \quad (x)(` \quad r \quad 1 \quad y)+O \text{ by (3.17)} \\ &= ` \quad (jQj \quad y)+r \quad y+(x+y)(jQj \quad y) \quad x \quad `+x \quad r+x \quad y+x+O \\ &\quad ` \quad (jQj \quad y)+r \quad y+(x+y)(jQj \quad y) \quad x \quad `+x \quad r+x \quad y+O \text{ since } x \quad 0 \\ &= ` \quad (jQj \quad y)+(r)(x+y)+(x+y)(jQj \quad y) \quad x \quad `+x \quad y+O \\ &\quad ` \quad (jQj \quad y)+(r)(x+y)+(x+y)(jQj \quad y) \quad x \quad `+O \text{ since } x \quad y \quad 0 \\ &= ` \quad (jQj \quad y \quad x)+(r)(x+y)+(x+y)(jQj \quad y)+O \\ &= ` \quad (jQj \quad (y+x))+(r)(x+y)+(x+y)(jQj \quad y)+O \\ &\quad ` \quad (jQj \quad (y+x))+(r)(x+y)+(x+y)(jQj \quad (y+x))+O \\ &= c \ (j+x): \end{aligned}$$

If instead jQj is odd:

$$\begin{aligned}
c\ (j) &= \ `\ \ (jQj\quad y) + r\ \ y + (y)(jQj\quad y) + O \\
&= \ `\ \ (jQj\quad y) + r\ \ y + (y)(jQj\quad y) + (x)(jQj\quad y)\quad (x)(jQj\quad y) + O \\
&= \ `\ \ (jQj\quad y) + r\ \ y + (x+y)(jQj\quad y)\quad (x)(jQj\quad y) + O \\
&\quad \ `\ \ (jQj\quad y) + r\ \ y + (x+y)(jQj\quad y)\quad (x)(`\quad r\quad y) + O \text{ by (3.20)} \\
&= \ `\ \ (jQj\quad y) + r\ \ y + (x+y)(jQj\quad y)\quad x\ \ `+x\ \ r+x\ \ y + O \\
&= \ `\ \ (jQj\quad y) + (r)(x+y) + (x+y)(jQj\quad y)\quad x\ \ `+x\ \ y + O \\
&\quad \ `\ \ (jQj\quad y) + (r)(x+y) + (x+y)(jQj\quad y)\quad x\ \ `+O \text{ since } x\ \ y\quad 0 \\
&= \ `\ \ (jQj\quad y\quad x) + (r)(x+y) + (x+y)(jQj\quad y) + O \\
&= \ `\ \ (jQj\quad (y+x)) + (r)(x+y) + (x+y)(jQj\quad y) + O \\
&\quad \ `\ \ (jQj\quad (y+x)) + (r)(x+y) + (x+y)(jQj\quad (y+x)) + O \\
&= c\ (j+x)\text{:}
\end{aligned}$$

Therefore, the cut at $c\ (j)$ is at least as large as the cut between any two vertices of Q after the first y vertices of Q and the cut after the last vertex of Q.

We also need to show that no cut between vertices of Q before $c\ (j)$ or at location $i\quad 1$ is larger than $c\ (j)$.

Consider the difference between $c\ (z\quad 1)$ and $c\ (z)$ for $1\quad z\quad y$, i.e., $c\ (z\quad 1)\quad c\ (z)$:

$$\begin{aligned}
c\ (z\quad 1)\quad c\ (z) &= r + \ `\ + O\quad (z(jQj\quad z)\quad (z\quad 1)(jQj\quad (z\quad 1)) + O) \\
&= r + \ `\ \ (zjQj\quad z^2\quad (z\quad 1)(jQj\quad z+1)) \\
&= r + \ `\ \ (zjQj\quad z^2\quad (zjQj\quad z^2+z\quad jQj+z\quad 1) \\
&= r + \ `\ \ (zjQj\quad z^2\quad (zjQj\quad z^2+2z\quad jQj\quad 1) \\
&= r + \ `\ \ (zjQj\quad z^2\quad zjQj+z^2\quad 2z+jQj+1) \\
&= r + \ `\ \ (jQj\quad 2z+1) \\
&= r + \ `\ \ jQj+2z\quad 1
\end{aligned}$$

We show that $r+\ `\ \ jQj+2y\quad 1\quad 0$. Suppose to the contrary that $r+\ `\ \ jQj+2y\quad 1 >$

0; then,

$$\begin{aligned} r + \ell - |Q| + 2y - 1 > 0 \\ \Rightarrow \ell - r + 2y > |Q| + 1 \\ \Rightarrow 2y > |Q| + 1 - \ell + r \\ \Rightarrow y > (|Q| + 1 - \ell + r)/2 \\ \Rightarrow y > |Q|/2 + (r - \ell + 1)/2 \\ \Rightarrow y > \lfloor |Q|/2 \rfloor + \lceil (r - \ell)/2 \rceil. \end{aligned}$$

However, by definition $y = \lfloor |Q|/2 \rfloor + \lceil (r - \ell)/2 \rceil$, so we have a contradiction.

Therefore, there is no cut between vertices of Q before $c(j)$ or at location $i - 1$ that is larger than $c(j)$. □

Theorem 3.7.5. Cutwidth *is fixed-parameter tractable when parameterized by the restricted twin cover number of the graph.*

Proof. Let G be a graph with $rtc(G) = |T| + q = k$ for some $q, k - 0$ and twin cover T. We can compute T in time $f(k) \cdot n^{O(1)}$ by Theorem 2.5.19. There are q non-trivial components of $G - T$. We modify the approach of ILP 3.7.3 as follows.

First, instead of trying every permutation of a vertex cover, we try every permutation of the twin cover T and q non-trivial components; there are $(|T|+q)!$ such permutations. Let $Q_1, \ldots, Q_q$ be the non-trivial components of $G - T$, and let $Q = \bigcup_{i=1}^{q} Q_i$. By Lemma 2.5.2, each non-trivial component of $G - C$ is an equivalence class of true twins; by Theorem 3.5.1 there is a cutwidth-minimal ordering of G such that each equivalence class of true twins is consecutive; we will find such an ordering using an ILP.

Let $\sigma_1, \ldots, \sigma_k$ be the ordering imposed on $T \cup (\bigcup_{i=1}^{q} Q_i)$ where σ_i is either some $t \in T$ or an arbitrary consecutive ordering of Q_j for some $1 - j - q$. Let $T_i = \bigcup_{j=1}^{i} V(\sigma_j)$, where $V(\sigma_i) = \sigma_i$ if $\sigma_i \in T$ (i.e., σ_i is a vertex in the twin cover), or $V(\sigma_i) = V(Q_j)$ for some $1 - i - q$ (i.e., σ_i is a non-trivial component of $G - T$).

We redefine a *location* as being between two vertices of the twin cover, between two non-trivial components of $G - T$, or between a non-trivial component of $G - T$ and a vertex of the twin cover. That is, a vertex $v \in V(G - (T \cup Q))$ is in L_0 if $v < \sigma_1$ or is in L_i if i is the largest integer such that $\sigma_i < v$ (the same as in the vertex cover case). This means the definition of "location" applies to "between twin cover vertices or non-trivial components" rather than "between vertex cover vertices" in the proof that follows.

For a set S and index i, the constant e_S^i and variable x_S^i are the same as in the bounded vertex cover number ILP formulation. That is, the constant e_S^i is the rank of a vertex with neighbourhood S at location i and the variable x_S^i is the number of vertices with neighbourhood S at location i. Note that since vertices of Q_i are true twins (Lemma 2.5.2), a vertex in $G \setminus (T \cup Q)$ is either adjacent to all of Q_i or none of it. Thus, there are still only $2^{rtc(G)}$ subsets S that we need to consider.

We now describe how to define the constant define e_i. If $\pi_i \in T$, then we define $e_i = |N(\pi_i) \setminus (T \cap T_i)| - |N(\pi_i) \setminus T_i|$ (also essentially the same as in the vertex cover case). Otherwise, $\pi_i = Q_j$ for some $1 \leq j \leq q$, and we define $e_i = |Q_j| - |N(\pi_i) \setminus (T_{|T|+q} \cap T_i)| - |Q_j| - |N(\pi_i) \setminus T_i|$ We can compute e_i for $1 \leq i \leq k$ before creating an instance of an integer linear program in time that only depends on k (we don't need to look at a component Q_j except to see its size during this process).

For each permutation, we create an instance of an integer linear program. We have the same objective function as in the vertex cover case, and keep Constraints 3.4 and 3.8.

As in the bounded vertex cover number case, a maximum cut may occur immediately after or before a vertex in the twin cover, so we keep Constraints 3.6 and 3.7. However, it may also be immediately before or after a consecutive set of vertices in some non-trivial component of $G \setminus T$, or in between two vertices of such a component.

Thus, we add the following constraint which counts the maximum cut when at least one side of the cut is a vertex of Q_i. The new variables are defined after the constraint is listed, and it is explained in the following claim.

$$c \leq f_i + \ell_i + \sum_{j=0}^{i-1} y_j + \sum_{j=0}^{i-1} \sum_{S \subseteq C} e_S^j \cdot x_S^j \quad \forall i \in \{1, \ldots, k\} \tag{3.21}$$

We introduce constants f_i and l_i for $1 \leq i \leq k$ to handle the above case. For π_i where $\pi_i = Q_j$ for some $1 \leq j \leq q$, we define the constant $\ell_i = |Q_j| - |N(\pi_i) \setminus T_i|$. For π_i where $\pi_i \in T$, we define $\ell_i = 0$. The constant ℓ_i represents the number of edges from π_i to its left, immediately before the first vertex of π_i, i.e., $E(V(\pi_i); T_{i-1})$ for $1 < i \leq |T| + q$. For $\pi_i \in T$, $f_i = 0$. For $\pi_i = Q_j$ for some $1 \leq i \leq q$, we let f_i be the value of the largest cut such that at least one vertex on either side of the cut is in π_i.

Claim 3.7.6. *Constraint 3.21 ensures that the objective function accounts for cuts in the orderings where the cut has at least one vertex of some Q_i on either side of it, and we can compute f_i constant time.*

Proof of claim: First, note that if $q = 0$ (i.e., T is a vertex cover of G), then Constraint 3.21 is exactly the same as Constraint 3.6. This follows from the fact that $f_i = \grave{}_i = 0$ for $_i$ when $_i \in T$.

In particular, the terms

$$\sum_{j=0}^{i-1} y_j + \sum_{j=0}^{i-1} \sum_{S \subseteq C} e_S^j \, x_S^j \quad \forall i \in \{1,\ldots,k\}$$

are the same as Constraint 3.6. As the discussion after the ILP Formulation 3.7.3 states, Constraint 3.6 counts the number of edges immediately left of a vertex in the vertex cover. Therefore, these terms count the edges immediately left of the vertices of a non-trivial component Q_i.

Thus, if we subtract $\grave{}_i$ from this value, we have the edges crossing over the vertices of the non-trivial component Q_i. This is because $\grave{}_i$ counts the edges from Q_i to the vertices of T_k that are left of Q_i, and any vertices of I to the left of Q_i are not adjacent to Q_i (as otherwise they would be in the component). Therefore, the terms

$$\grave{}_i + \sum_{j=0}^{i-1} y_j + \sum_{j=0}^{i-1} \sum_{S \subseteq C} e_S^j \, x_S^j \quad \forall i \in \{1,\ldots,k\}$$

are the edges over Q_i (the value O in the proof of Lemma 3.7.4). We can compute f_i in this case by using Lemma 3.7.4 if $|N(\ _i) \setminus T_i| \quad |N(\ _i) \setminus (T \cap T_i)|$ or applying Lemma 3.7.4 to ^{R} otherwise. Lemma 3.7.4 indicates which index should be used to compute the largest cut; let $$ be the index provided by the lemma. Note that once an ordering $_1,\ldots,\ _k$ is known, the values of $\grave{}$ and r required by the lemma are known for a $_j = Q_i$, as $N(Q_i) \quad T$. Suppose that $$ vertices of Q_i are to the left of $$ and $$ vertices are to its right (so that $+ \ = |Q_i|$). Since each Q_i is a set of true twins, we can compute f_i as follows

$$f_i = \grave{} \quad + r \quad + \quad ;$$

since $\grave{}$ represents the edges from the left of Q_i to the vertices of Q_i on the right of the cut, r represents the edges from the right of Q_i to the vertices of Q_i on the left of the cut, and $$ represents the edges between vertices of Q_i. Since we know which index to check, we can compute the value f_i in constant time after the ordering $_1,\ldots,\ _k$ has been chosen.

The correctness of the ILP follows from the previous claim and the discussion prior to it.

Finally, since we have only added $2k$ more constants and the variables in the ILP remain bounded, Theorem 3.7.2 implies that $\mathsf{Cutwidth}$ is fixed-parameter tractable with the parameter $rtc(G)$. □

Chapter 4

Imbalance

4.1 Introduction

The Imbalance problem was introduced by Biedl et al. [8] in the context of graph drawing, where such an ordering is helpful (Kant [85], Kant and He [86], Papakostas and Tollis [117], Wood [137, 138]).

Biedl et al. [8] showed that Imbalance is NP-complete for bipartite graphs with degree at most 6 and weighted trees and they provided a pseudo-polynomial time algorithm for weighted trees which runs in linear time on unweighted trees. Kára et al. [87] showed that the problem is NP-complete for graphs of degree at most 4 and planar graphs.

In the parameterized complexity setting, Imbalance has been studied almost as well as Cutwidth. Fellows et al. [52] showed that the problem is FPT when the parameter is the vertex cover number of the graph. Bakken [4] showed that Imbalance is FPT when parameterized by the neighbourhood diversity of the graph. Lokshtanov et al. [104] showed that Imbalance is FPT when parameterized by the solution size k by constructing an algorithm that runs in time $O(2^{O(k \log k)} \quad n^{O(1)})$, or when parameterized by $k = f(tw(G); \Delta(G))$ for some function f where $tw(G)$ is the treewidth of the graph and $\Delta(G)$ is the maximum degree of the graph. The maximum degree is likely required as the problem is similar to Cutwidth, and that problem is NP-complete for graphs with treewidth at most 2 (Monien and Sudborough [113]). Previously, we claimed (Gorzny and Buss [68]) that Imbalance is FPT when parameterized by twin cover number of the graph. However, Misra and Mittal [111] showed that our approach was not correct (and this is explained in Section 4.8). Misra and Mittal [111] also show that Imbalance is FPT

when parameterized by the size of a twin cover and the size of the largest set of true twins outside the twin cover and show that there is an XP-time Imbalance when the parameter is the twin cover number. This XP time algorithm (an algorithm with run time $O(n^{f(k)})$) is weaker than an FPT (an algorithm with run time $O(f(k)n^{O(1)})$) result (and the XP result does not imply an FPT result). Gaspers et al. [61] showed that Imbalance is equivalent to the Graph Cleaning problem, which yielded a $O(n^{bk=2c}(n+m))$ time parameterized algorithm where k is the solution size.

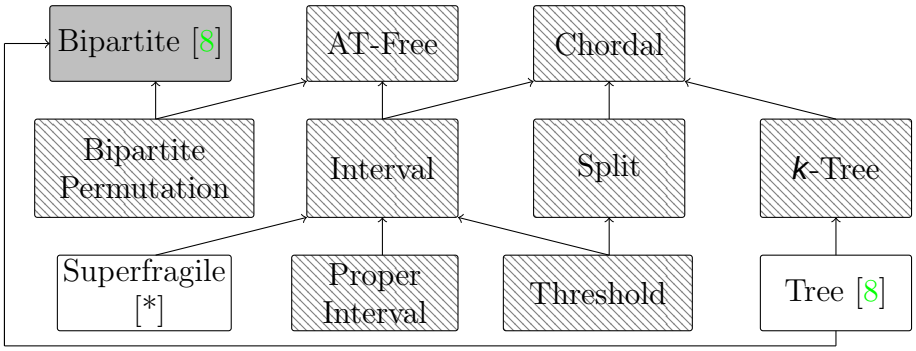
Some of the known complexity results for Imbalance are shown in Figure 4.1.

We aim to show that the complexity of Imbalance is often exactly the same as the complexity of Cutwidth. This is not always true (and there is no equivalence between the problems): for example, the FPT results for the parameter neighbourhood diversity are different for Cutwidth and Imbalance. The order in which we prove things may suggest that Imbalance is a specialized case of Cutwidth. However, we do not know how to formalize this specialization, but think it would be of interest and enable many results to carry over from one problem to the next without effort like that of this thesis. The orderings that minimize Cutwidth also appear to be similar to those for Imbalance. In the case of a kernel for graphs with bounded vertex cover number, the exact same reduction for Cutwidth is immediately applicable. A dynamic programming approach similar to the Cutwidth algorithm also apples to the imbalance case with only a few changes. For many proofs used for Imbalance results, counting arguments are the only tools at our disposal, and many are only possible because of the structure we impose on the graphs in question. More general arguments appear necessary to prove results for larger classes of graphs.

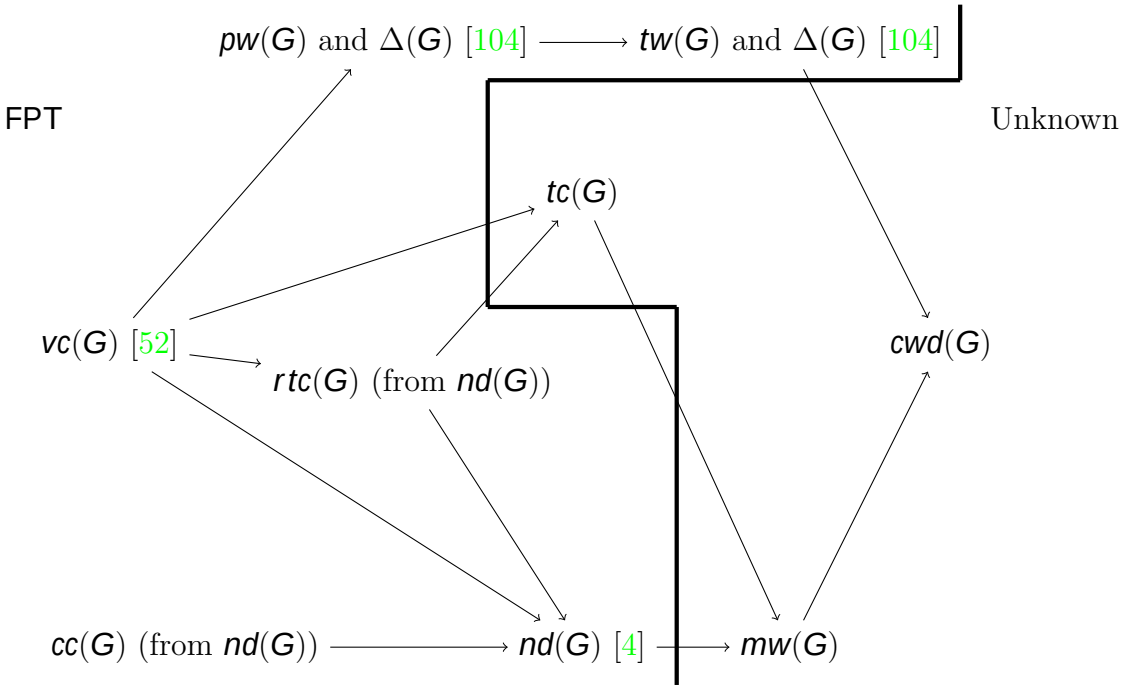
Summary of Results

In this chapter we first show some preliminary results for Imbalance. These include observations that bound the difference in imbalances after swapping the location of some sets in an ordering (Section 4.2), and proving closed formulas for Imbalance on some very small graph classes (Section 4.3). We then show some helpful theorems that show there are two useful structures for imbalance-minimal orderings on arbitrary graphs: one with sets of true twins appearing together, and one that says an independent set within a graph can always be perfectly balanced (Section 4.4).

Afterwards, we establish some complexity results for Imbalance. In the parameterized complexity setting, we show that existing techniques from Cutwidth enable a faster FPT algorithm for Imbalance when the parameter is the vertex cover number of the



(a) Some known complexity results for Imbalance on restricted graph classes. The problem is NP-complete for classes with a solid gray background, has unknown complexity for classes with a hatched background, and is in P otherwise. Results for classes marked with [*] are shown in this work.



(b) Some known parameterized complexity results for Imbalance for common graph parameters. For each parameter to the left of the thick line, there is an FPT algorithm for Imbalance with that parameter. The complexity of Imbalance is open for those parameters to the right of the thick line.

Figure 4.1: Some known complexity results for Imbalance.

graph and that Imbalance is not likely to admit a polynomial-size kernel with the same parameter (Section 4.7). Specifically, we show an improved FPT algorithm for graphs with bounded vertex cover number that is an improvement over the existing $O(2^{2^{O(vc(G))}}n^{O(1)})$ time algorithm of Fellows et al. [52]; our algorithm runs in time $O(2^{vc(G)}n^{O(1)})$. Since $vc(G)$ is $O(n=2)$ for a bipartite graph, this algorithm is also an improvement over the existing $O(\frac{n}{2}!)$ algorithm for bipartite graphs (Biedl et al. [8]). Unlike for Cutwidth, we do not show that Imbalance is FPT for the parameters restricted twin cover number and edge clique cover number of the graph, as this is implied by the algorithm for graphs with bounded neighbourhood diversity (Bakken [4]). In Section 4.5, we prove that Imbalance is in P for superfragile graphs. In all cases, the complexity results for Imbalance that we establish match those of Cutwidth.

These results may help solve larger open questions. For example, the result for superfragile graphs builds towards an understanding of the Imbalance problem on general interval graphs, for which the complexity of the problem is unknown.

4.2 Preliminaries

In this section we establish some useful results related to Imbalance.

We start with observations related to the reverse of an ordering. Unlike in the case of cutwidth, reversing an ordering has no effect on the imbalances of each individual vertex.

Observation 4.2.1. *If is an ordering of a graph with its reverse ordering ^{R}, then $im(\ ) = im(\ ^R)$.*

Observation 4.2.2. *If is an ordering of a graph with its reverse ordering ^{R}, then $(v) = {}_R(v)$ for all vertices v of .*

We now prove some helpful and straightforward lemmas.

Lemma 4.2.3. *Let $G = (V; E)$ be a graph. Suppose that rank (x) 0 for some vertex x 2 V and ordering of G. Let ^{0} be an ordering of G such that $_{Vnfxg} = {}^0_{Vnfxg}$. If rank $_0(x) =$ rank (x) 2y for some y 1, then $_0(x) =$ $(x) + 2y$.*

Proof. Suppose that

$$0 \quad \text{rank } (x) \quad \text{rank } (x) \quad 2y = \text{rank } {}_0(x):$$

Then

$$\text{rank}\ (x) = j\text{rank}\ (x)j \quad 0; \tag{4.1}$$

and

$$(\text{rank}\ (x) \quad 2y) = \quad \text{rank}\ {}_0(x) = j\text{rank}\ {}_0(x)j = \quad \text{rank}\ (x) + 2y \quad 0: \tag{4.2}$$

Putting it all together, we have

$$\begin{aligned}
{}_0(x) &= j\text{rank}\ {}_0(x)j \\
&= \ j\text{rank}\ {}_0(x)j \\
&= j \quad \text{rank}\ (x) + 2yj \text{ by } (4.2) \\
&= \ j\text{rank}\ (x)j + 2y \ \text{ by } (4.1) \\
&= j\text{rank}\ (x)j + 2y \text{ since } y \quad 1 \\
&= \quad (x) + 2y;
\end{aligned}$$

as required. □

Lemma 4.2.4. *Suppose that X is a set of twins which are consecutive in some ordering .*

1. *If jYj neighbours of X are moved forward past X to the right of X to obtain ^{0}, then ${}_0(X) \quad (X) + 2 \ jXj \ jYj$.*
2. *If jYj neighbours of X are moved backward past X to the left of X to obtain ^{0}, then ${}_0(X) \quad (X) + 2 \ jXj \ jYj$.*

Proof. We prove only the first statement, as the second is symmetric (apply the first statement to ^{R}).

Note that

$$\begin{aligned}
\text{rank}\ {}_0(x) &= \text{succ}\ {}_0(x) \quad \text{pred}\ {}_0(x) \\
&= (\text{succ}\ (x) \quad jYj) \quad (\text{pred}\ (x) + jYj) \\
&= \text{succ}\ (x) \quad \text{pred}\ (x) \quad 2jYj \\
&= \text{rank}\ (x) \quad 2jYj:
\end{aligned}$$

Therefore, each vertex $x \in X$ can have its imbalance increase by at most $2|Y|$. Thus, if $\pi_X = \pi'_X = \langle x_0, x_1, \ldots, x_k \rangle$,

$$\begin{aligned}
\sigma'(X) &= \sum_{i=0}^{k} \sigma'(x_i) \\
&\leq \sum_{i=0}^{k} \left(\sigma(x_i) + 2|Y| \right) \\
&= \sum_{i=0}^{k} \sigma(x_i) + 2|Y||X| \\
&= \sigma(X) + 2|Y||X|,
\end{aligned}$$

as required. □

Lemma 4.2.5. *Let $G = (V, E)$ be a graph. Suppose that $y > \text{rank}(x) \geq 0$ for some $y \geq 1$, vertex $x \in V$, and ordering π of G. Let π' be an ordering of G such that $\pi_{V \setminus \{x\}} = \pi'_{V \setminus \{x\}}$. If $\text{rank}_{\pi'}(x) = \text{rank}_{\pi}(x) \geq 2y$, then $\sigma'(x) > \sigma(x)$.*

Proof. Assume that $\text{rank}_{\pi'}(x) = \text{rank}_{\pi}(x) \geq 2y$ and $y > \text{rank}_{\pi}(x) \geq 0$. Then we have:

$$0 \leq \text{rank}_{\pi}(x) < y \tag{4.3}$$

$$\Longrightarrow 2y \leq \text{rank}_{\pi}(x) + 2y < y$$

$$\Longrightarrow 2y \leq \text{rank}_{\pi'}(x) < y \text{ by assumption.} \tag{4.4}$$

Multiplying (4.3) by -1 yields:

$$0 \geq \text{rank}_{\pi}(x) > -y \tag{4.5}$$

Combining (4.4) with (4.5) yields

$$0 \geq \text{rank}_{\pi}(x) > -y > \text{rank}_{\pi'}(x).$$

Since $0 \geq \text{rank}_{\pi}(x) > \text{rank}_{\pi'}(x)$, $\sigma'(x) = |\text{rank}_{\pi'}(x)| \geq |\text{rank}_{\pi}(x)| > \sigma(x)$, as required. □

Lemma 4.2.6. *Let $G = (V, E)$ be a graph. Suppose that $\text{rank}_{\pi}(x) > y > 0$ for some $y \geq 1$, vertex $x \in V$, and ordering π of G. Let π' be an ordering of G such that $\pi_{V \setminus \{x\}} = \pi'_{V \setminus \{x\}}$. If $\text{rank}_{\pi'}(x) = \text{rank}_{\pi}(x) \geq 2y$, then $\sigma'(x) < \sigma(x)$.*

Proof. Assume that rank $_{\prime}(x) =$ rank (x) $2y$ and rank $(x) > y > 0$. Then we have:

$$\begin{aligned} & \text{rank } (x) > y && > 0 && (4.6)\\ \Longrightarrow\ & \text{rank } (x) \quad 2y > \quad y && > \quad 2y && \\ \Longrightarrow\ & \text{rank } _{\prime}(x) > \quad y && > \quad 2y \text{ by assumption:} && (4.7) \end{aligned}$$

We have two cases based on the sign of rank $_{\prime}(x)$.

Case 1: rank $_{\prime}(x) < 0$. Combining the case assumption along with (4.7) yields:

$$0 > \text{rank } _{\prime}(x) > \quad jYj \Longrightarrow \quad 0 < \quad \text{rank } _{\prime}(x) < jYj;$$

and therefore, when combined with (4.6), we have

$$\text{rank } (x) > y > \quad \text{rank } _{\prime}(x) > 0;$$

which implies that $_{\prime}(x) < \quad (x)$.

Case 2: rank $_{\prime}(x) \quad 0$. Since rank $(x) >$ rank $(x) \quad 2y =$ rank $_{\prime}(x) \quad 0$, we have that $_{\prime}(x) < \quad (x)$. □

Lemma 4.2.7. *Let $G = (V; E)$ be a graph. Suppose that rank $(x) \quad y > 0$ for some $y \quad 1$, vertex $x \ 2 \ V$, and ordering $\quad$ of G. Let ^{0} be an ordering of G such that $_{Vnfxg} = \ ^{0}_{Vnfxg}$. If rank $_{\prime}(x) =$ rank $(x) \quad 2y$ and rank $_{\prime}(x) < 0$, then $_{\prime}(x) \quad (x)$.*

Proof. Assume that rank $(x) \quad y > 0$ for some $y \quad 1$, rank $_{\prime}(x) =$ rank $(x) \quad 2y$ and rank $_{\prime}(x) < 0$. These assumptions yield the following observation:

$$\begin{aligned} & 0 > \text{rank } _{\prime}(x) \text{ by assumption} && \\ \Longrightarrow\ & 0 > \text{rank } _{\prime}(x) = \text{rank } (x) \quad 2y \text{ by assumption} && \\ \Longrightarrow\ & 2y > \text{rank } _{\prime}(x) = \text{rank } (x): && (4.8) \end{aligned}$$

Combining (4.8) with the assumption rank $(x) \quad y > 0$ yields

$$2y > \text{rank } (x) \quad y > 0: \tag{4.9}$$

Multiplying (4.9) by $\quad 1$, we get

$$2y < \quad \text{rank } (x) \quad y < 0: \tag{4.10}$$

Subtracting $2y$ from (4.9), we get

$$0 > \text{rank } (x) \quad 2y \quad y > \quad 2y: \tag{4.11}$$

Now we can combine (4.10) and (4.11), along with the fact rank $_0(x) =$ rank (x) $2y$ that to get

$$0 > \text{rank } (x) \quad 2y = \text{rank } _0(x) \quad y \quad \text{rank } (x) \quad 2y:$$

in which case $0 >$ rank $_0(x)$ rank (x), which implies that $_0(x)$ (x). □

The next lemma is an analogue of Lemma 3.2.11, but deals with the imbalance of an ordering.

Lemma 4.2.8. *If moving $v_p =$ (p) backward to a position q results in an ordering ^{0} such that rank $_0(v_p)$ 0, then $im(\ ^0)$ $im(\ )$. If moving $v_p =$ (p) forward to a position q results in an ordering ^{0} such that rank $_0(v_p)$ 0, then $im(\ ^0)$ $im(\ )$.*

Proof. Let $v_p =$ (p). If moving v_p forward results in rank $_0(v_p)$ 0, then by definition of the rank of a vertex, it must have been the case that rank (v_p) 0 as well. Similarly, if moving v_p backwards results in rank $_0(v_p)$ 0, then by definition of the rank of a vertex, it must have been the case that rank (v_p) 0 as well. We therefore proceed by cases based on the sign of rank (v_p).

Case 1: rank (v_p) 0. Let be obtained by moving v_p forward past $v_{p+1} =$ $(p+1)$ where (v_p) 0.

- If v_{p+1} 2 $N(v_p)$, then neither v_p nor v_{p+1} has its imbalance change. Therefore, $im(\ )$ $im(\ )$.
- Otherwise, v_{p+1} 2 $N(v_p)$. Then $(v_p) =$ (v_p) 2 0 (which implies that $(v_p) >$ 1) and (v_{p+1}) $(v_{p+1}) + 2$. Therefore, $im(\ )$ $im(\ )$.

We can repeat this as long as v_p moves forward and the resulting position has a non-negative rank. In this case, $im(\ )$ $im(\ )$, as required.

Case 2: rank (v_p) 0. Let be obtained by moving v_p backward past $v_{p\ 1} =$ $(p$ 1) where (v_p) 0.

- If $v_{p\ 1}$ 2 $N(v_p)$, then neither v_p nor $v_{p\ 1}$ has its imbalance change. Therefore, $im(\ )$ $im(\ )$.
- Otherwise, $v_{p\ 1}$ 2 $N(v_p)$. Then $(v_p) =$ (v_p) 2 0 (which implies that (v_p) 2) and $(v_{p\ 1})$ $(v_{p\ 1}) + 2$. Therefore, $im(\ )$ $im(\ )$.

We can repeat this as long as v_p moves backward and the resulting position has a non-positive rank. In this case, $im(\ ) \quad im(\ )$, as required. □

Lemma 4.2.8 above does not describe all the scenarios where vertices can be freely moved without increasing the imbalance of an ordering. For example, a pendant vertex with negative rank may freely be moved forward without increasing the imbalance of the ordering, as it is not adjacent to any vertex to its right in the ordering.

Lemma 4.2.9. *Let $G = (V; E)$ be a graph and let $(u; v)$ 2 E for $u; v$ 2 V. If $u <\ v$ in an ordering of G, then*

1. *rank $_v(u) =$ rank $(u) \quad 1$, and*
2. *rank $_u(v) =$ rank $(v) + 1$.*

Proof. We first prove (1). In v, which is obtained by deleting v from , u has one less neighbour to its right but the same amount to its left. From the definition of rank, we have

$$\begin{aligned} \text{rank}\ _v(u) &= \text{succ}\ _v(u) \quad \text{pred}\ _v(u) \\ &= \text{succ}\ (u) \quad 1 \quad \text{pred}\ (u) \\ &= \text{rank}\ (u) \quad 1. \end{aligned}$$

We now prove (2). In u, which is obtained by deleting u from , v has one less neighbour to its left but the same amount to its right. From the definition of rank, we have

$$\begin{aligned} \text{rank}\ _u(v) &= \text{succ}\ _u(v) \quad \text{pred}\ _u(v) \\ &= \text{succ}\ (v) \quad (\text{pred}\ (v) \quad 1) \\ &= \text{rank}\ (v) + 1. \end{aligned}$$

□

Lemma 4.2.10. *Let G be a graph. If there is an imbalance-minimal ordering of G where $(1) = v$, then $im(G) \quad im(G \quad v)$.*

Proof. Let be an imbalance-minimal ordering of G where $(1) = v$. Since $(1) = v$, all of $N(v)$ is to the right of v in , i.e., $(v) = jN(v)j = \text{rank}\ (v) \quad 0$.

Let $R^{\ 0} = fu\ 2\ N(v)\ j\ \text{rank}\ (u)\quad 0g$ and $R^{<0} = fu\ 2\ N(v)\ j\ \text{rank}\ (u) < 0g$. The sets $R^{\ 0}$ and $R^{<0}$ partition $N(v)$, that is, $jR^{\ 0}j + jR^{<0}j = jN(v)j$.

If $u\ 2\ R^{\ 0}$, then rank $_v(u) =$ rank $(u) + 1$, and therefore $_v(u) = (u) + 1$ by Lemma 4.2.9. If $u\ 2\ R^{<0}$, then rank $_v(u) =$ rank $(u) + 1$, and therefore $_v(u) = (u)\quad 1$ by Lemma 4.2.9.

Putting it all together,

$$
\begin{aligned}
im(G) &= im(\ ) \\
&= \sum_{u2V} (u) \\
&= \sum_{u2V nfvg} (u) + (v) \\
&= \sum_{u2V nfvg} (u) + jN(v)j \\
&= \sum_{u2V nfvg} (u) + jR^{\ 0}(v)j + jR^{<0}(v)j \\
&= \sum_{u2V nfvg} (u) + jR^{\ 0}(v)j \quad jR^{<0}(v)j \text{ since } jR^{<0}(v)j \quad 0 \\
&= \sum_{u2V(G)nN(v)} (u) + \sum_{u2R^{\ 0}} (u) + \sum_{u2R^{<0}} (u) \\
&\quad + jR^{\ 0}(v)j \quad jR^{<0}(v)j \\
&= \sum_{u2V(G)nN(v)} (u) + \sum_{u2R^{\ 0}} (u) + jR^{\ 0}(v)j \\
&\quad + \sum_{u2R^{<0}} (u) \quad jR^{<0}(v)j \\
&= \sum_{u2V(G)nN(v)} (u) + \sum_{u2R^{\ 0}} (\ \ (u) + 1) \\
&\quad + \sum_{u2R^{<0}} (\ \ (u) \quad 1) \\
&= \sum_{u2V(G)nN(v)} {}_v(u) + \sum_{u2R^{\ 0}} {}_v(u) + \sum_{u2R^{<0}} {}_v(u) \\
&= \sum_{u2V(G)nfvg} {}_v(u)
\end{aligned}
$$

$= im(\quad v)$
$\quad im(G \quad v)$:

□

Let X be a consecutive set in an ordering $\quad$. Recall that $\quad(X)$ denotes the sum of the imbalance of vertices in X for the ordering $\quad$. For an ordering $\quad$ and a consecutive set of true twins vertices X, we will use $\hat{\ }(X)$ to denote $|\text{succ}(X) \quad \text{pred}(X)|$.

Observation 4.2.11. *Let G be a graph and let X be an equivalence class of true twins. Suppose that $\quad$ and $\quad$ are two orderings of G that place X consecutively. If $\hat{\ }(X) > \hat{\ }(X)$, then $\quad(X) > \quad(X)$.*

Recall that $\quad_X$ denotes the ordering of X imposed by $\quad$, for a set X. The notation $\quad_{<X}$, where X is a consecutive set in $\quad$, denotes the ordering of $\quad$ restricted to the vertices prior to the consecutive set X (we will use analogous for $>$, $\quad$, and $\quad$).

Lemma 4.2.12. *Let G be a graph with $|U(G)| \quad 1$. If $\quad$ is an ordering of a G that places $U(G)$ consecutively and rank $(u) \quad 0$ for all $u \in U(G)$, then $|\quad_{<U(G)}| \quad |\quad_{>U(G)}|$.*

Proof. Write $\quad = hL \quad U(G) \quad Ri$ for some sets $L; R \quad (V(G) \setminus U(G))$. Let $u \in U(G)$ be rightmost in $U(G)$ among all vertices of $U(G)$. Then,

$$\begin{aligned} 0 \quad \text{rank}(u) &= \text{succ}(u) \quad \text{pred}(u) \\ &= |R| \quad (|L| + |U(G)| \quad 1) \\ &= |R| \quad |L| \quad |U(G)| + 1 \\ &\quad |R| \quad |L| \quad |U(G)| + |U(G)| \\ &= |R| \quad |L|; \end{aligned}$$

that is, $|R| \quad |L| \quad 0$. Since $R = \quad_{<U(G)}$ and $L = \quad_{>U(G)}$, we are done. □

The next lemma provides bounds on the difference $|im(\ ') \quad im(\ )|$ when $\ '$ is obtained by interchanging two consecutive sets of twins in $\quad$. Figure 4.2 illustrates example cases for the lemma.

Lemma 4.2.13. *Let X be a set of true twins with $Y \quad N(X)$ for some set of vertices Y. Suppose that $X < Y$ for some $\quad$, where there is no vertex between X and Y, and X and Y are consecutive in $\quad$. Let $\ '$ be obtained from $\quad$ by swapping the positions of X and Y in $\quad$.*

If pred $(X) \quad$ succ (X), then $\quad(X) < \quad_0(X) \quad (X) + 2 \quad |X| \quad |Y|$. Otherwise, pred $(X) <$ succ (X) and

1. *if* succ (X) $jYj <$ *pred* $(X) <$ *succ* (X), *then* $(X)+2\,jXj\,jYj$ $_0(X) >$ (X);
2. *else if* *pred* $(X) =$ *succ* (X) jYj, *then* $_0(X) =$ (X);
3. *otherwise* *pred* $(X) <$ *succ* (X) jYj, *and* $_0(X) <$ (X).

Proof. We break the proof up into several claims. The following equations are necessary, and arise due to the construction of ^{0} from $$:

$$\text{succ}\ (X) = \text{succ}\ _0(X) + jYj \tag{4.12}$$

$$\text{pred}\ (X) = \text{pred}\ _0(X) \quad jYj \tag{4.13}$$

By definition, we have the following

$$\begin{aligned}\text{succ}\ (X) \quad \text{pred}\ (X) &= \text{succ}\ _0(X) + jYj \quad \text{pred}\ (X) \text{ by (4.12)}\\ &= \text{succ}\ _0(X) + jYj \quad (\text{pred}\ _0(X) \quad jYj) \text{ by (4.13)}\\ &= \text{succ}\ _0(X) \quad \text{pred}\ _0(X) + 2jYj \text{ by (4.13)};\end{aligned}$$

which implies

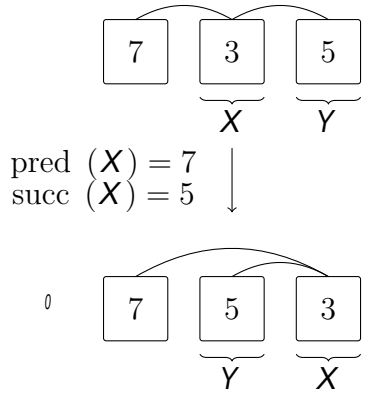
$$\text{succ}\ (X) \quad \text{pred}\ (X) \quad 2jYj = \text{succ}\ _0(X) \quad \text{pred}\ _0(X): \tag{4.14}$$

Claim 4.2.14. *If* *pred* (X) $\neq$ *succ* (X), *then* $(X) <$ $_0(X)$ $(X) + 2$ jXj jYj.

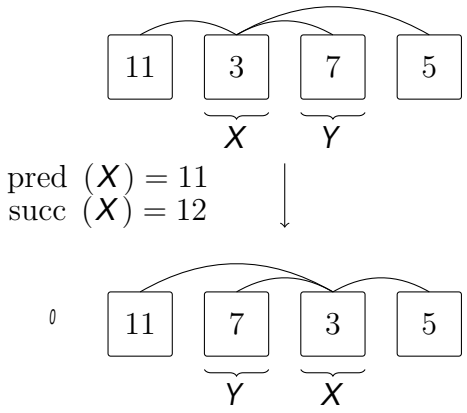
Proof of claim: Note that

$$\begin{aligned}0 \quad \hat{} \ (C_1) &= j\text{succ}\ (X) \quad \text{pred}\ (X)j\\ &= j\text{pred}\ (X) \quad \text{succ}\ (X)j\\ &= \text{pred}\ (X) \quad \text{succ}\ (X) \text{ since pred } (X) \quad \text{succ}\ (X)\\ &< \text{pred}\ (X) + jYj \quad \text{succ}\ (X) + jYj \text{ since } jYj \quad 1\\ &= \text{pred}\ _0(X) \quad (\text{succ}\ (X) \quad jYj) \text{ by (4.13)}\\ &= \text{pred}\ _0(X) \quad \text{succ}\ _0(X) \text{ by (4.12)}\\ &= j\text{pred}\ _0(X) \quad \text{succ}\ _0(X)j\\ &= \hat{} \ _0(C_1):\end{aligned}$$

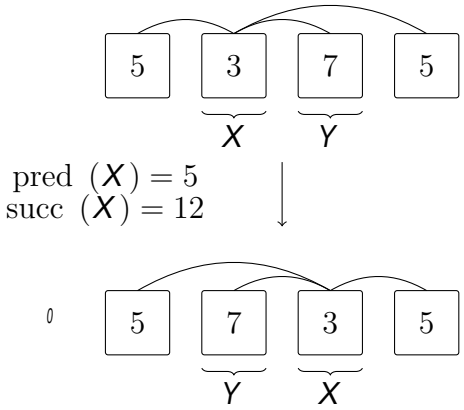
Therefore, we have that $(X) <$ $_0(X)$ by Observation 4.2.11. We also have that $_0(X)$ $(X) + 2$ jXj jYj by Lemma 4.2.4. Combining these two facts concludes the proof of the claim.



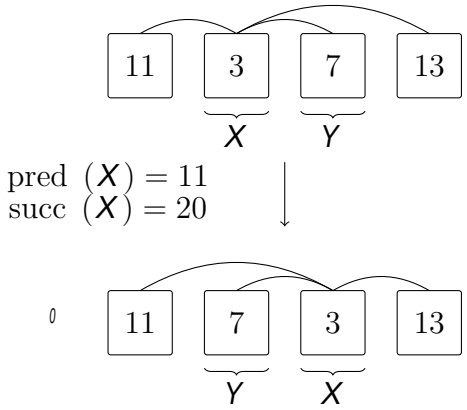
(a) An example of statement 1 where pred (X) succ (X). Observe that $(X) = 6$ and $_0(X) = 36 = (X) + 2\ jXj\ jYj$.



(b) An example of statement 1a) where pred (X) < succ (X) and succ (X) jYj < pred (X) < succ (X). Observe that $(X) = 5$ and $(X) + 2\ jXj\ jYj = 47 > {}_0(X) = 39$.



(c) An example of statement 1b) where pred (X) < succ (X) and pred (X) = succ (X) jYj. Observe that $(X) = 21 = {}_0(X)$.



(d) An example of statement 1c) where pred (X) < succ (X) and pred (X) < succ (X) jYj. Observe that $(X) = 27$ and $_0(X) = 15$.

Figure 4.2: An illustration of Lemma 4.2.13. An edge between boxes indicates that every vertex in one box is adjacent to every vertex in the other box. Each of the sets X and Y are a set of true twins in these examples.

Claim 4.2.15. *Suppose that pred* $(X) <$ *succ* (X). *If succ* (X) $jYj <$ *pred* $(X) <$ *succ* (X), *then* $(X) + 2$ jXj jYj $_0(X) >$ (X).

Proof of claim: Since pred $(X) <$ succ (X), $\hat{}$ $(X) > 0$, and moreover,

$$\begin{aligned} 0 < \hat{} (X) &= j\text{succ } (X) \quad \text{pred } (X)j \\ &= \text{succ } (X) \quad \text{pred } (X) \text{ since pred } (X) < \text{succ } (X) \\ &< \text{succ } (X) \quad (\text{succ } (X) \quad jYj) \text{ since succ } (X) \quad jYj < \text{pred } (X) \\ &= jYj: \end{aligned}$$

Thus we have

$$\begin{aligned} & 0 < \text{succ } (X) \quad \text{pred } (X) && < jYj && (4.15) \\ =) \quad & 2jYj < \text{succ } (X) \quad \text{pred } (X) \quad 2jYj && < \quad jYj \\ =) \quad & 2jYj < \text{succ } {}_0(X) \quad \text{pred } {}_0(X) && < \quad jYj \text{ by (4.14)} && (4.16) \end{aligned}$$

Now multiply 4.15 by $\quad 1$ to get

$$0 > \quad (\text{succ } (X) \quad \text{pred } (X)) > \quad jYj: \tag{4.17}$$

Combine (4.16) and (4.17) to get

$$0 > \quad (\text{succ } (X) \quad \text{pred } (X)) > \quad jYj > \text{succ } {}_0(X) \quad \text{pred } {}_0(X) > \quad 2jYj:$$

Since $0 >$ $(\text{succ } (X)$ $\text{pred } (X)) >$ $\text{succ } {}_0(X)$ $\text{pred } {}_0(X)$, $\hat{}$ ${}_0(X) = j\text{succ } {}_0(X)$ $\text{pred } {}_0(X)j > j$ $(\text{succ } (X)$ $\text{pred } (X))j =$ $\hat{}$ (X).

Therefore, we have that $\quad (X) <$ $\quad {}_0(X)$ by Observation 4.2.11. We also have that ${}_0(X)$ $\quad (X) + 2$ jXj jYj by Lemma 4.2.4. Combining these two facts concludes the proof of the claim.

Claim 4.2.16. *Suppose that pred* $(X) <$ *succ* (X). *If pred* $(X) =$ *succ* (X) jYj, *then* ${}_0(X) =$ (X).

Proof of claim: Note that

$$\begin{aligned}
0 \quad \hat{} \ (C_1) &= j\text{succ } (X) \quad \text{pred } (X)j \\
&= \text{succ } (X) \quad \text{pred } (X) \text{ since pred } (X) < \text{succ } (X) \\
&= \text{pred } (X) + jYj \quad \text{pred } (X) \text{ by claim assumption} \\
&= jYj \\
&= \text{succ } (X) \quad \text{succ } (X) + jYj \\
&= \text{succ } (X) \quad (\text{succ } (X) \quad jYj) \\
&= \text{succ } (X) \quad \text{pred } (X) \\
&= \text{succ } (X) \quad \text{pred } (X) + jYj \quad jYj \\
&= \text{succ }_0(X) \quad (\text{pred } (X) + jYj) \text{ by (4.12)} \\
&= \text{succ }_0(X) \quad \text{pred }_0(X) \text{ by (4.13)} \\
&= j\text{succ }_0(X) \quad \text{pred }_0(X)j \\
&= \hat{}\ _0(C_1);
\end{aligned}$$

implying that $\quad (X) = \quad _0(X)$, as required.

Claim 4.2.17. *Suppose that* pred (X) < succ (X). *If* pred (X) < succ $(X) \quad jYj$, *then* $_0(X) < \quad (X)$.

Proof of claim: First, note that

$$\text{succ }_0(X) + \text{pred }_0(X) = \text{succ } (X) + \text{pred } (X): \tag{4.18}$$

By the claim assumption and (4.12), we also have

$$\text{pred } (X) < \text{succ } (X) \quad jYj = \text{succ }_0(X): \tag{4.19}$$

Combining (4.18) and (4.19), we must also have that

$$\text{succ } (X) > \text{pred }_0(X): \tag{4.20}$$

Now observe that

$$\begin{aligned}
\hat{} \ (X) &= j\text{succ } (X) \quad \text{pred } (X)j \\
&= \text{succ } (X) \quad \text{pred } (X) \text{ since pred } (X) < \text{succ } (X) \\
&> \text{pred }_0(X) \quad \text{pred } (X) \text{ by (4.20)} \\
&= \text{pred } (X) + jYj \quad \text{pred } (X) \text{ by (4.12)} \\
&= jYj > 0:
\end{aligned} \tag{4.21}$$

There are two cases, based on whether pred $_0(X)$ succ $_0(X)$.

Case 1: pred $_0(X)$ succ $_0(X)$.

$$\begin{aligned} 0 \quad \hat{} \ _0(X) &= j\text{succ } _0(X) \quad \text{pred } _0(X)j \\ &= j\text{pred } _0(X) \quad \text{succ } _0(X)j \\ &= \text{pred } _0(X) \quad \text{succ } _0(X) \text{ by case assumption} \\ &< \text{pred } _0(X) \quad \text{pred } (X) \text{ by (4.19)} \\ &= \text{pred } (X) + jYj \quad \text{pred } (X) \text{ by (4.13)} \\ &= jYj: \end{aligned} \tag{4.22}$$

Combining (4.21) and (4.22), we have that $_0(X) <$ (X) by Observation 4.2.11, in this case.

Case 2: pred $_0(X) <$ succ $_0(X)$. We break this case into two subcases, based on how large $\hat{}$ (X) is.

Case 2A: $\hat{}$ (X) $2jYj$. In this case,

$$\begin{aligned} & \hat{} \ (X) \quad 2jYj \\ =) \quad & j\text{succ } (X) \quad \text{pred } (X)j \quad 2jYj \\ =) \quad & \text{succ } (X) \quad \text{pred } (X) \quad 2jYj \text{ since pred } (X) < \text{succ } (X) \\ =) \quad & \text{succ } (X) \quad \text{pred } (X) \quad 2jYj \quad 0 \text{ since } jYj \quad 0: \end{aligned}$$

Therefore, we have that $\hat{}$ $(X) =$ succ (X) pred (X) succ (X) pred (X) $2jYj =$ $\hat{}$ $_0(X)$ 0 by the case assumption. Therefore, we have that we have that $_0(X) <$ (X) by Observation 4.2.11, in this case.

Case 2B: $\hat{}$ $(X) < 2jYj$.

Note that

$$\begin{aligned} & 2jYj > \text{succ } (X) \quad \text{pred } (X) \quad > jYj \quad > 0 \text{ by (4.21)} \\ =) \quad & 0 > \text{succ } (X) \quad \text{pred } (X) \quad 2jYj \quad > \quad jYj > \quad 2jYj \\ =) \quad & 0 > \text{succ } _0(X) \quad \text{pred } _0(X) \quad > \quad jYj > \quad 2jYj \text{ by (4.14).} \end{aligned} \tag{4.23}$$

Now we can use (4.21):

$$\begin{aligned} & \hat{} \ (X) > jYj \quad > 0 \\ =) \quad & \text{succ } (X) \quad \text{pred } (X) > jYj \quad > 0 \text{ by (4.21)} \\ =) \quad & (\text{succ } (X) \quad \text{pred } (X)) < \quad jYj \quad < 0: \end{aligned} \tag{4.24}$$

Therefore, from 4.23 and 4.24, we have 0 > succ ◊(X) pred ◊(X) > jYj > (succ (X) pred (X)). This implies ^ ◊(X) = jsucc ◊(X) pred ◊(X)j < j (succ (X) pred (X))j = ^ (X). Therefore, we have that we have that ◊(X) < (X) by Observation 4.2.11, in this case.

Since we have ◊(X) < (X) in any case, the claim is proved.

The lemma follows from the previous claims.

□

4.3 Small Graph Classes

Lokshtanov et al. [104] proved Lemma 3.1.1 and wondered if it may be of general interest. In this section we show how Lemma 3.4.2, which strengthens Lemma 3.1.1, can be used to get closed formulas for Imbalance. We focus on some small graph classes that have simple formulas for Cutwidth.

The *triangulated triangle* T` is the graph[1] whose vertices are the non-negative integer triples with sum ` such that vertices (x;y;z) and (x⁰;y⁰;z⁰) are adjacent if and only if jx x⁰j + jy y⁰j + jz z⁰j = 2 (see, e.g., Hochberg et al. [81]). Lin et al. [100] show how such graphs can be drawn in the plane by putting vertex (x;y;z) at point (x;y). An example is provided in Figure 4.3a, which also indicates the position of each vertex in a cutwidth-minimal ordering. Lin et al. [100] determined the following closed formula for the cutwidth of these graphs, using the ordering indicated on Figure 4.3a.

Theorem 4.3.1 (Theorem 1.1, Lin et al. [100])**.** *cw*(T`) = 2`*.*

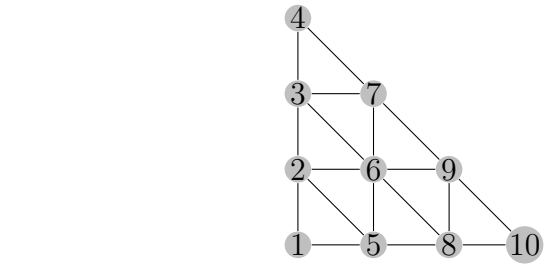
We prove the following corollary, establishing the imbalance of triangulated triangles.

Corollary 4.3.2. *im*(T`) = 2*cw*(T`) = 4`*.*

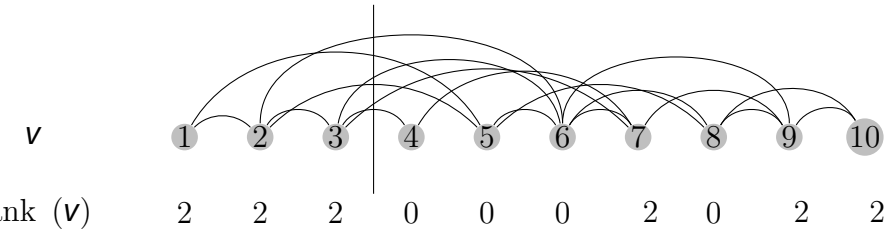
Proof. Given a triangulated triangle T`, draw it in the plane using the method of Lin et al. [100]. Let be an ordering of the vertices which corresponding to the lexicographic order of coordinates (x;y) which contain a vertex of a graph, starting with (0;0);(0;1);:::;(0;` + 1) (see e.g., Figure 4.3b). Lin et al. showed that such orderings are cutwidth-minimal.

In this ordering, ((1)) = rank ((1)) = 2 and ((i)) = rank ((i)) = 2 for all 1 < i < ` + 1. For any vertex v with coordinate (x;0) for x > 0 such that v is not last in

[1] Note that these graphs are not triangulated in the sense of chordal graphs.



(a) The graph T_3 drawn in the plane.



(b) An ordering such that $cw(\ ) = cw(T_3)$. The vertical line is the first largest cut.

Figure 4.3: An example triangulated triangle and its cutwidth-minimal ordering.

, $(v) = \text{rank}\ (v) = 0$, as it has two neighbours before it in the ordering and two after. For any vertex u with coordinate $(x; y)$ where $x > 0$ and there is no vertex at coordinate $(x; y')$ where $y' > y$, $(u) = 2$ and rank $(u) = \ 2$.

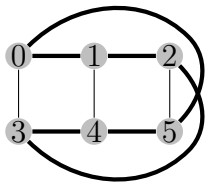
The only vertices which remain are vertices w with coordinates $(x; y)$ $x > 0$ such that they have neighbours at coordinates $(x; y+1)$, $(x+1; y)$, $(x+1; y \ \ 1)$, $(x; y \ \ 1)$, $(x \ \ 1; y)$, and $(x \ \ 1; y+1)$; half of these neighbours are placed after w in and the other half appear before w in .

It is easy to verify that the first largest cut is immediately after the first vertex u with coordinate $(0; `)$.

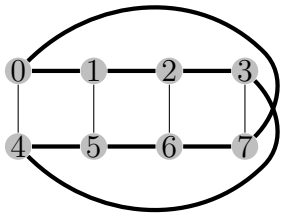
Thus, vertices only have non-negative rank before the first largest cut and vertices only have non-positive rank after it. Therefore by Lemma 3.4.2, $im(G) = 2cw(G) = 4`$. □

The *Möbius ladder* M_k $(k \ \ 3)$ is a cycle of length $2k$ with each pair of vertices at distance k apart in the cycle joined by an edge. Example Möbius ladders are provided in Figures 4.4a and 4.4b. Lin et al. [101] proved the following constant value for Möbius ladders.

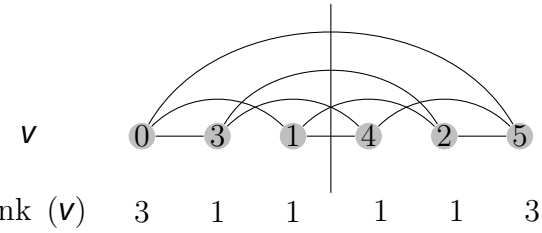
Theorem 4.3.3 (Theorem 4.2, Lin et al. [101])**.** $cw(M_k) = 5$.



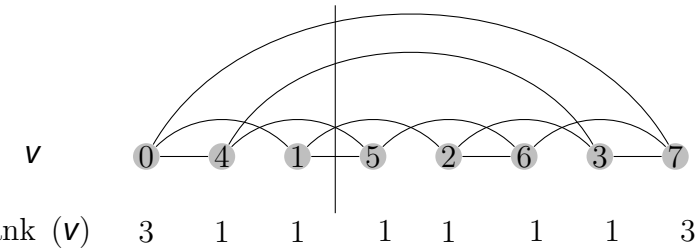
(a) The graph M_3. Thick edges are cycle edges.



(b) The graph M_4. Thick edges are cycle edges.



(c) The graph M_3 drawn to illustrate a cutwidth-minimal ordering. The vertical line indicates the (only) largest cut.



(d) The graph M_4 drawn to illustrate a cutwidth-minimal ordering. The vertical line indicates the first largest cut.

Figure 4.4: Example Möbius ladders and their cutwidth-minimal orderings.

In the next theorem, we prove that $im(M_3) = 2cw(M_3) = 10$, but for $k > 3$, $im(M_k) > 2cw(M_3)$. This provides an explicit class of graphs for which the bound provided by Lemma 3.1.1 is not tight. For $k > 3$, Lemma 3.4.2 is not applicable since there is no largest cut in a cutwidth-minimal ordering which results in a sign change for the ranks of the vertices, unlike when $k = 3$ (see Figures 4.4c and 4.4d).

Theorem 4.3.4. *$im(M_k) = 2k + 4$.*

Proof. Note that every vertex has odd degree exactly 3 (its two neighbours on the cycle and the single vertex "across" from it on the cycle). Therefore $\quad(\ (1)) = \quad(\ (n)) = 3$ and $\quad(\ (i)) \quad 1$ for any $1 < i < n$ for any imbalance-minimal ordering $\ $. Therefore if $im(\ ) = im(G)$, then $im(\ ) \quad (2k \quad 2)+6 = 2k+4$. Suppose that M_k was constructed from the cycle $fv_0; \ldots; v_{2k\ 1}g$. An ordering $\quad = hv_0; v_k; v_1; v_{k+1}; \ldots; v_{k\ 1}; v_{2k\ 1}i$ has imbalance $im(\ ) = 3 + 2(k \quad 1) + 3 = 2k + 4$, which is the best possible. □

Lemma 4.3.5. *If G is the complete graph K_n, then $im(G) = 2cw(G) = 2 \quad \frac{n}{2} \quad \frac{n}{2}$.*

Proof. Since all vertices are true twins in a complete graph, any ordering $\quad$ of G achieves is both cutwidth-minimal and imbalance-minimal. The result follows from Lemma 3.3.1 and Lemma 3.4.2. □

Lemma 4.3.6. *Consider a complete bipartite graph $K_{m;n}$. Then*

$$im(K_{m;n}) = \begin{cases} mn & \text{if } mn \text{ is even} \\ mn + 1 & \text{if } mn \text{ is odd} \end{cases}$$

Proof. The result follows from Lemma 3.3.2 and Lemma 3.4.2. Hence $im(G) = 2cw(G)$ in this case. □

There may be other small graph classes for which closed formulas for imbalance can be proved using Lemmas 3.4.2 or 3.1.1. As we will see in Section 4.7, these lemmas can also be used in more involved settings.

4.4 Structures for Optimal Orderings

In this chapter we show that, for any graph, two different types of imbalance-optimal orderings always exist.

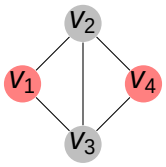


Figure 4.5: An example graph G where it is not possible to perfectly balance a given independent set and have each equivalence class of true twins be consecutive. Vertices 1 and 4 form an independent set, while vertices 2 and 3 are true twins. An imbalance-minimal ordering of G balancing $\{v_1, v_4\}$ (satisfying Theorem 4.4.1) is $\langle v_2, v_1, v_4, v_3 \rangle$, while an imbalance-minimal ordering with $\{v_2, v_3\}$ (satisfying Theorem 4.4.2) is $\langle v_1, v_2, v_3, v_4 \rangle$. It is easy to see that in any ordering of G such that $(v_1) = 0$, the ordering necessarily places v_1 between v_2 and v_3. Similarly, any ordering ^{0} of G that keeps the vertices v_2 and v_3 consecutive places v_1 on one side of $\{v_2, v_3\}$, in which case $^0(v_1) = 2$.

The first type of optimal ordering shows that an independent set can always be perfectly balanced in some imbalance-minimal ordering.

The second shows that there are imbalance-minimal orderings that group each equivalence class of true twins consecutively (similar to those for Cutwidth provided by Theorem 3.5.1). This type of ordering is helpful for superfragile graphs (Section 4.5).

Note that it is not always possible to have imbalance-minimal orderings for general graphs which satisfy both constraints at once; see Figure 4.5 for an example.

The first structure is easier to establish, and is formalized in the following theorem.

Theorem 4.4.1. *Let $G = (A \cup B, E)$ be a graph where B is an independent set and let be such that $im(\ ) = im(G)$. There exists an imbalance-minimal layout such that for all $b \in B$, $(b) \in \{0, 1\}$; moreover, $_A$ agrees with $_A$.*

Proof. Let be an imbalance-minimal ordering and suppose that $b \in B$ has $(b) > 1$. Without loss of generality, suppose that b has $k \geq 2$ more neighbours to its right than its left. Let w be the vertex to the right of b in , and let ^{0} be the result of swapping b and w. If $(b, w) \notin E$, the swap does not change the total imbalance of the ordering as $^0(b) = (b)$ and $^0(w) = (w)$. If $(b, w) \in E$, $^0(b) = (b) - 2$ and $^0(w) = (w) + 2$. In this case, k is reduced by 2 (since the number of its neighbours on its left increases by one and the number of its neighbours on its right decreased by one). We can repeat this until $k = 1$

and $(b) = 1$ if $|N(b)|$ is odd, or $k = 0$ and $(b) = 0$ if $|N(b)|$ is even. Since we did not move any vertices of A past any other vertices in A, $_A$ agrees with $_A$. □

We now prove that there are imbalance-minimal orderings where the equivalence classes of true twins of a graph are each consecutive. The proof follows the same structure as the proof of Theorem 3.5.1.

Theorem 4.4.2. *For any graph G, there exists an imbalance-minimal ordering of $V(G)$ such that each equivalence class of true twins appears consecutively in .*

Recall that ( ) is the number of maximal consecutive sets of true twins in the ordering , and that $tt(G)$ is the number of equivalence classes of true twins in the graph G. Lastly, recall that by definition, ( ) $tt(G)$ for any graph G and ordering of G.

Proof of Theorem 4.4.2. Say that if and only if ( ) < ( ) and $im(\ ) \quad im(\ )$ for two orderings and of G.

Suppose that is such that $im(\ ) = im(G) = k$ but ( ) > $tt(G)$ (if ( ) = $tt(G)$ and $im(\ ) = im(G)$ there is nothing to prove). We will find an ordering . Repeating this until the resulting order is such that ( ) = $tt(G)$ proves the theorem.

Let $U \quad V(G)$ be an equivalence class of true twins which is not consecutive in . We may assume that there are at least two distinct, maximal consecutive sets of vertices $U_1 \quad U$ and $U_2 \quad U$ in . Pick two such sets U_1 and U_2 such that $U_1 < U_2$ and there is no vertex $u \in U$ such that $U_1 < u < U_2$. Let $u \in U_1$ be the rightmost vertex of U_1 in , and $v \in U_2$ be the leftmost vertex of U_2 in . We have that $u < v$, and that u is not beside v in (as otherwise U_1 would be beside U_2 and $U_1 \cup U_2$ would be a consecutive set in).

Write $= hL \quad U_1 \quad Y \quad U_2 \quad Ri$ for some set Y and (possibly empty) sets L and R. Furthermore, partition Y into maximal consecutive sets $Y_1, \ldots, Y_\ell$ where each Y_j ($1 \quad j \quad \ell$) is a set of true twins from the same equivalence class. Since U_1 is not beside U_2 in , $\ell \quad 1$.

We may assume $Y_\ell \quad N(v)$ as otherwise we can swap U_2 left until it is immediately to the right of one of its neighbours. Similarly $Y_1 \quad N(u)$, as otherwise we can swap U_1 right until it is immediately to the left of one of its neighbours.

We will show that can be modified to an ordering where U_1 is beside U_2 (so that $U_1 \cup U_2$ is consecutive) such that the resulting order has no worse imbalance and does not have any set Y_i split either. The imbalance of vertices in $L \cup R$ will not change.

We consider cases based on the signs of rank (u) and rank (v). Since u and v are true twins, by Observation 3.2.10, the case where rank (u) 0 but rank (v) 0 is impossible.

Case 1: rank (u) 0 and rank (v) 0. If we move v immediately right of u to get ^{0}, v would have rank at most rank (u) 2 0 by Lemma 3.2.9. Since the rank of a vertex increases by two when it moves backward past one of its neighbours, and v moved past jM_vj such neighbours in ^{0}, we must have had rank (u) 1 $(2$ $jM_vj)$. Since the ranks of true twins are non-increasing (Observation 3.2.10), all vertices in U_2 have rank at most 1 $(2$ $jM_vj)$. Since the rank of u 2 U_2 was negative and increased to a value which is still negative, the imbalance of u 2 U_2 has improved by 2 jM_vj. Therefore, if we let $= hL$ U_1 U_2 Y Ri, we have that $(U_2) =$ (U_2) 2 jU_2j jM_vj. By Lemma 4.2.4, (M_v) $(Y) + 2$ jU_2j jM_vj. Note that $(Y \cap M_v) =$ $(Y \cap M_v)$ since those vertices have the same number of vertices on either side of them in both orderings. Thus, $im(\ )$ $im(\ )$, and since $U_1 \cup U_2$ is consecutive, $(\ ) <$ $(\ )$; therefore $\ $. We have an ordering $\ $ in the case rank (u) 0 and rank (v) 0.

Case 2: rank $(u) > 0$ and rank (v) 0. If we move u immediately left of v to get ^{0}, u would have rank at least rank $(v) + 2$ 0 by Lemma 3.2.9. Since the rank of a vertex decreases by two when it moves forward past one of its neighbours, and v moved past jM_vj such neighbours in ^{0}, we must have had rank (u) 2 jM_vj. Since the ranks of true twins are non-increasing (Observation 3.2.10), all vertices in U_1 have rank at least 2 jM_vj. Since the rank of u 2 U_1 was positive and decreased to a value which is still non-negative, the imbalance of u 2 U_1 has improved by 2 jM_vj. Therefore, if we let $= hL$ Y U_1 U_2 Ri, we have that $(U_1) =$ (U_1) 2 jU_1j jM_vj. By Lemma 4.2.4, (M_v) $(Y) + 2$ jU_1j jM_vj. Note that $(Y \cap M_v) =$ $(Y \cap M_v)$ since those vertices have the same number of vertices on either side of them in both orderings. Thus, $im(\ )$ $im(\ )$, and since $U_1 \cup U_2$ is consecutive, $(\ ) <$ $(\ )$; therefore $\ $. We have an ordering $\ $ in the case rank $(u) > 0$ and rank (v) 0.

Case 3: rank $(u) > 0$ and rank $(v) < 0$. There is a rightmost position between u and v such that u will be perfectly balanced (and the rank of v will be non-negative). We can move u forward to that position, followed by all of U_1, by Lemma 4.2.8. Call the resulting ordering ^{0}. Then, the position immediately right of u in ^{0} is where v achieves is maximal non-positive rank, and would have an imbalance that is no worse than (v). Therefore, we can move v backward to that position, followed by all of U_2, by Lemma 4.2.8. Call the resulting ordering ^{1}.

We have reduced $\ $ by 1 by gathering the sets U_1 and U_2, but this may have resulted in splitting some set Y_j $(1$ j $`)$, which may have increased $\ $ by 1. We may assume that $(\ ^{1}) =$ $(\ )$, as otherwise $(\ ^{1}) <$ $(\ )$ and $im(\ ^{1})$ $im(\ )$; therefore ^{0} $\ $ and

we are done.

Therefore, we must have that $^1 = hL^0, Y_j^0\ U_1\ U_2\ Y_j^{00}\ R^0 i$ where Y_j^0 and Y_j^{00} partition Y_j, $L^0 = L\ [\ \bigcup_{i=1}^{j-1} Y_i$, and $R^0 = R\ [\ \bigcup_{i=j+1} Y_i$. By Theorem 3.4.4 and the fact that $U_1\ [\ U_2$ and Y_j are sets of true twins, there is an ordering where $^0 = hL^0\ Y_j\ U_1\ [\ U_2\ R^0 i$ or $^0 = hL^0\ U_1\ [\ U_2\ Y_j\ R^0 i$ and $im(\ ^0)\ \ im(\ ^1)$. Thus, $(\ ^0) = (\ ^1)\ \ 1$ and $im(\ ^0)\ \ im(\ ^1)\ \ im(\ )$ and ^{0} I , as required. We have an ordering ^{0} I in the case rank $(u) > 0$ and rank $(v) < 0$.

Since we have found an ordering I in every case, the proof is complete. □

Consequences

The remaining results in this section follow from the previous theorem. They are not used in this thesis, but may be helpful for future proofs. In particular, they may enable inductive proofs in some settings. The first result establishes a condition for when a vertex is first in some imbalance-minimal ordering.

Corollary 4.4.3. *Let G be a graph such that $U(G)$ 6= ;. If v 2 $V(G)$ n $U(G)$ is such that $N(v) = U(G)$, then there is an imbalance-minimal ordering of G such that $(1) = v$ and $U(G)$ is consecutive.*

Proof. Let be an imbalance-minimal ordering provided by Theorem 4.4.2, i.e., one which places all of $U(G)$ consecutively. Suppose without loss of generality that $v <\ U(G)$ (as otherwise we can use ^{R}).

Since v is only adjacent to $U(G)$, v is not adjacent to the vertex u immediately left of v in . We can swap the positions of u and v without increasing the imbalance, as neither vertex has its rank change in this operation. We can repeatedly swap v with the vertex to its left in the resulting ordering until v is the first vertex, and call that ordering . No swap increased the imbalance of the ordering, so $im(\ ) = im(\ ) = im(G)$. □

The previous corollary allows us to say a bit more about the structure of (optimal) orderings, as shown in the next two lemmas.

Lemma 4.4.4. *Let G be a graph such that $U(G)$ 6= ; and let v 2 $V(G)$ n $U(G)$ be such that $N(v) = U(G)$. If $im(G) = im(G\ \ v)$, then there is an imbalance-minimal ordering of G such that rank $(u)\ \ 0$ for all u 2 $U(G)$.*

Proof. The proof is by contradiction; suppose to the contrary that $im(G) = im(G \quad v)$ and, for every imbalance-minimal ordering , there is at least one vertex of $N(v)$ such that rank $(u) < 0$. Let be a imbalance-minimal ordering of G where $(1) = v$; one exists by Corollary 4.4.3. Since $(1) = v$, all of $N(v)$ is to the right of v in , i.e., $(v) = jN(v)j = \text{rank } (v) > 0$.

Let $R^{\ 0} = fu \ 2 \ N(v) \ j \text{ rank } (u) \quad 0g$ and $R^{<0} = fu \ 2 \ N(v) \ j \text{ rank } (u) < 0g$. The sets $R^{\ 0}$ and $R^{<0}$ partition $N(v)$, that is, $jR^{\ 0}j + jR^{<0}j = jN(v)j$. Since every imbalance-minimal ordering of G has at least one vertex of $N(v)$ with negative rank, $jR^{<0}j \quad 1$.

If $u \ 2 \ R^{\ 0}$, then rank $_v(u) = \text{rank } (u) + 1$, and therefore $_v(u) = (u) + 1$ by Lemma 4.2.9. If $u \ 2 \ R^{<0}$, then rank $_v(u) = \text{rank } (u) + 1$, and therefore $_v(u) = (u) \quad 1$ by Lemma 4.2.9.

Putting it all together,

$$
\begin{aligned}
im(G) &= im(\ ) \\
&= \sum_{u2V} (u) \\
&= \sum_{u2V nfvg} (u) + (v) \\
&= \sum_{u2V nfvg} (u) + jN(v)j \\
&= \sum_{u2V nfvg} (u) + jR^{\ 0}(v)j + jR^{<0}(v)j \\
&> \sum_{u2V nfvg} (u) + jR^{\ 0}(v)j \text{ since } jR^{<0}(v)j \quad 1 \\
&> \sum_{u2V nfvg} (u) + jR^{\ 0}(v)j \quad jR^{<0}(v)j \text{ since } jR^{<0}(v)j \quad 1 \\
&= \sum_{u2V(G)nN(v)} (u) + \sum_{u2R^{\ 0}} (u) + \sum_{u2R^{<0}} (u) \\
&\quad + jR^{\ 0}(v)j \quad jR^{<0}(v)j \\
&= \sum_{u2V(G)nN(v)} (u) + \sum_{u2R^{\ 0}} (u) + jR^{\ 0}(v)j \\
&\quad + \sum_{u2R^{<0}} (u) \quad jR^{<0}(v)j
\end{aligned}
$$

$$
\begin{aligned}
&= \sum_{u \in V(G) \setminus N(v)} (u) + \sum_{u \in R^{0}} (\quad (u) + 1) \\
&\quad + \sum_{u \in R^{<0}} (\quad (u) \quad 1) \\
&= \sum_{u \in V(G) \setminus N(v)} {}_{v}(u) + \sum_{u \in R^{0}} {}_{v}(u) + \sum_{u \in R^{<0}} {}_{v}(u) \\
&= \sum_{u \in V(G) \setminus \{v\}} {}_{v}(u) \\
&= im(\quad v) \\
&\quad im(G \quad v) \\
&= im(G) \text{ by assumption on } v;
\end{aligned}
$$

which is a contradiction. □

Lemma 4.4.5. *Let G be a graph such that $U(G) \neq \emptyset$; and let $v \in V(G) \setminus U(G)$ be such that $N(v) = U(G)$. If there is an ordering $\quad$ of G such that rank $(u) \quad 0$ for all $u \in U(G)$ and $\quad(1) = v$, then $im(\ ) = im(\quad v)$.*

Proof. Since $\quad(1) = v$, all of $N(v)$ is to the right of v in $\quad$, i.e., $\quad(v) = |N(v)| = \text{rank } (v) > 0$.

For all $u \in U(G)$, rank $\quad_{v}(u) = \text{rank } (u) + 1$, and therefore $\quad_{v}(u) = \quad (u) + 1$ by Lemma 4.2.9.

Putting it all together,

$$
\begin{aligned}
im(G) &= im(\ ) \\
&= \sum_{u \in V} (u) \\
&= \sum_{u \in V \setminus \{v\}} (u) + \quad (v) \\
&= \sum_{u \in V \setminus \{v\}} (u) + |N(v)| \\
&= \sum_{u \in V(G) \setminus N(v)} (u) + \sum_{u \in U(G)} (u) + |N(v)| \\
&= \sum_{u \in V(G) \setminus N(v)} (u) + \sum_{u \in U(G)} (\quad (u) + 1)
\end{aligned}
$$

$$
\begin{aligned}
&= \sum_{u \in V(G) \setminus N(v)} \sigma_v(u) + \sum_{u \in U(G)} \sigma_v(u) \\
&= \sum_{u \in V(G) \setminus \{v\}} \sigma_v(u) \\
&= \mathit{im}(\quad v);
\end{aligned}
$$

as required. □

4.5 Superfragile Graphs

Superfragile graphs are a subclass of interval graphs for which the complexity of Cutwidth is known to be $O(n^2)$ (Lilleeng [99]). We now show that the complexity of Imbalance is the same on this class of graphs.

The result uses a solution to the following problem, Number Partitioning.

Problem 4.5.1 (Number Partitioning). *Given a list $a_1; a_2; \ldots; a_N$ of positive integers, find a subset $A \quad \{1; \ldots; N\}$ minimizing the discrepancy*

$$\left| \sum_{i \in A} a_i \quad \sum_{i \notin A} a_i \right| :$$

Number Partitioning is NP-complete when the values a_i are unbounded, but has a polynomial time solution when these values are bounded (see, e.g., Mertens [109]). That is, Number Partitioning has a pseudo-polynomial time algorithm.

Finally, recall that by Observation 2.3.1, if G is a connected superfragile graph which is not a complete graph, then G was constructed by a complete join of U and the graph consisting of the cliques $C_1; \ldots; C_k$. The $k+1$ equivalence classes of true twins in G are the sets $C_1; \ldots; C_k$ along with the set U.

We are now ready to prove the main result of this section.

Theorem 4.5.2. *If G is a superfragile graph, then $\mathit{im}(G)$ can be computed in $O(n^2)$ time.*

Proof. We may assume G is connected; if not, we can compute the imbalance of each component and add them together. If G is a complete graph, the result follows by Theorem 4.3.5. By definition of a superfragile graph, if G is not a clique, the set $U(G)$ of universal vertices is a clique, and $G \quad U(G)$ is a disjoint union of cliques $C_1; \ldots; C_k$ (Observation

2.3.1). Each C_i is an equivalence class of true twins in G. Therefore by Theorem 4.4.2 there is an imbalance-minimal ordering where the vertices of each C_i appear consecutively. Similarly, $U(G)$ is an equivalence class of true twins (Observation 2.1.1) and as such $U(G)$ would appear together in one such as well. Since $N(C_i) = U(G)$ for any i, the only set which we need to minimize the imbalance of is $U(G)$, as necessarily C_i will be entirely on one side of $U(G)$ in .

Finding can be done by creating an instance of the Number Partitioning problem. Set $a_i = jC_ij$ for 1 i k. Observe that a_i $n = jV(G)$ for all 1 i k and that $k < n$ since $U(G)$ is not empty. An $O(N \quad A)$-time algorithm is known for this problem (see, e.g., Mertens [109]), where N is the size of the list and A bounds each element in the list. Taking $N = A = n$ means we can solve this instance in $O(n^2)$ time. Given a solution A, we can create an imbalance-minimal ordering as follows. Starting with an empty ordering, for each a_i 2 A, append all the vertices of C_i in any order to the existing ordering. Then, append all the vertices of $U(G)$ in any order. Finally, for each a_i 2 A, append all the vertices of C_i consecutively. Take the resulting order to be : each equivalence class of true twins is consecutive, and the imbalance of $U(G)$ is minimized. Thus this ordering has the minimum imbalance over all orderings which place each equivalence class of twins consecutively and is an imbalance-minimal ordering of G. □

4.6 Proper Interval k-Trees

In this section, we show that proper interval graphs that are also k-trees have imbalance equal to exactly twice their cutwidth.

An ordering of a graph G is called a *regular ordering* (or *regular labelling*) of G if for every edge $(u;v)$ 2 $E(G)$ with $u < v$, $V(u;v) = fx$ 2 $V(G)$ j (u) (x) $(v)g$ is a clique of G. Such an ordering is also called a *unit interval* ordering (Corneil and Stacho [38]).

Graphs which have a regular ordering are precisely the proper interval graphs (Looges and Olariu [105]). Yuan and Zhou [140] showed that regular orderings are optimal for several linear layout problems including Cutwidth and Optimal Linear Arrangement. Using LBFS$^+$ (Algorithm 2.6.3), it is possible to recognize if a graph is a proper interval graph and generate a regular ordering if it is.

Theorem 4.6.1 (Theorems 2 and 9, Corneil [31])**.** *A graph G is a proper interval graph if and only if the third LBFS$^+$ sweep on G is a regular ordering.*

We prove the results of this section in the context of regular orderings which are obtained from LBFS and LBFS$^+$. Recall that LBFS$^+$ is LBFS which breaks ties based on the position of tied vertices in another ordering provided as input; every LBFS$^+$-generated ordering is therefore an LBFS-generated ordering. Corneil et al. [36] proved the following characterization of end-vertices of a regular orderings generated by LBFS. Recall that a vertex x is *simplicial* if its neighbourhood induces a complete graph, and *admissible* if it does not have any pair of vertices which are unrelated with respect to x. We extend this to additionally characterize the first vertex in a regular ordering by observing that LBFS$^+$ on such an ordering reverses it.

Theorem 4.6.2 (Theorem 4.5, Corneil et al. [36])**.** *If σ is an ordering of an interval graph generated by LBFS, then $\sigma(n)$ is simplicial and admissible.*

Theorem 4.6.3 (Theorem 5, Charbit et al. [24])**.** *If G is a proper interval graph with regular ordering σ, then LBFS$^+(\sigma) = \sigma^R$.*

Corollary 4.6.4. *If σ is a regular ordering of a proper interval graph, then both σ and σ^R are LBFS orderings. Moreover, both $\sigma(1)$ and $\sigma(n)$ are simplicial and admissible vertices.*

Proof. Let σ be a regular ordering of a proper interval graph. By Theorem 4.6.3, LBFS$^+(\sigma) = \sigma^R$. Therefore, we have LBFS$^+$(LBFS$^+(\sigma)$) $= (\sigma^R)^R = \sigma$. Combining this with Theorem 4.6.2 proves the corollary. □

Therefore, $N[\sigma(1)]$ and $N[\sigma(n)]$ are cliques as $\sigma(1)$ and $\sigma(n)$ are both simplicial. In fact, these closed neighbourhoods must form maximal cliques: if there is another clique C containing, say without loss of generality $N[\sigma(1)]$, then any vertex of $C \setminus N[\sigma(1)]$ must also be adjacent to every vertex of $N[\sigma(1)]$, and in particular, it must also be adjacent to $\sigma(1)$. We formalize this as the following observation so that we can reference it.

Observation 4.6.5. *If σ is a regular ordering of a proper interval graph, $N[\sigma(1)]$ and $N[\sigma(n)]$ are maximal cliques in G.*

Before proceeding, recall the following well-known characterization of proper interval graphs, where a $K_{1,3}$ is also called a claw graph.

Theorem 4.6.6 (e.g., Theorem 1, Gardi [60])**.** *For an undirected graph G, the following statements are equivalent:*

1. *G is a proper interval graph.*

2. G is an interval graph that is $K_{1,3}$-free (i.e., G does not have an induced $K_{1,3}$).

Recall that a k-tree is a graph which can be constructed by starting with a complete graph on $k+1$ vertices and repeatedly adding vertices which are adjacent to k vertices which form a clique. As a result of this construction, no k-tree has a clique of size greater than $k+1$, and k-trees are chordal.

For a pair (n,k) where $k < n$, we construct proper interval k-trees as follows. Starting with an ordering of n vertices, for each $1 \quad i \quad n$, add the set $V(\ (i);\ (j))$ as a clique, where $j = \min\{i+k, n\}$. By construction, this ordering is regular and therefore these graphs are proper interval graphs by Theorem 4.6.1. An alternative construction of the same ordering starts with an ordering of $k+1$ vertices which form a clique, and repeatedly appends a vertex which is adjacent to the last k vertices of the previous ordering, until the length of the ordering is n. Therefore, these graphs are also k-trees. Moreover, for each pair (n,k) where $k < n$, there is only one graph G which is obtained by the previous construction. Examples are shown in Figure 4.6.

Before establishing the imbalance of these graphs, we show that they are edge-maximal graphs on n vertices for a fixed pathwidth value. That is, it is not possible to take such a graph and add an edge without also increasing its pathwidth. Moreover, this shows that these graphs are unique for a given pair (n,k), regardless of construction.

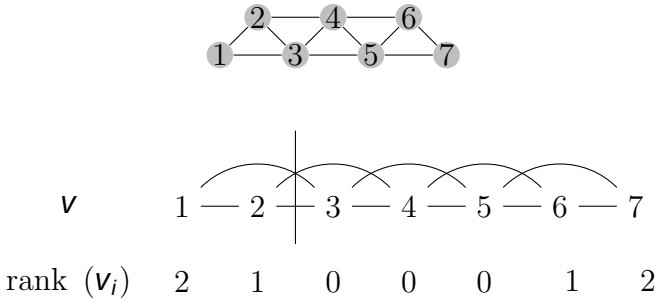
Recall that a graph has pathwidth k if the vertices can be put into bags with size at most $k+1$, every edge has both endpoints in some bag, and the bags can be linearly ordered so that the bags containing each vertex are consecutive.

Lemma 4.6.7 (Lemma 5.4, Proskurowski and Telle [120])**.** *A proper interval graph G with pathwidth k is edge-maximal if and only if G has an interval model corresponding to a path decomposition $X_1, \ldots, X_m$ where (1) $|X_i| = k+1$ for all $1 \quad i \quad m$, (2) $|X_i \setminus X_{i+1}| = k$ for $1 \quad i < m$, and (3) each $v \in V(G) \setminus (X_1 \cup X_m)$ is in at least $k+1$ bags.*

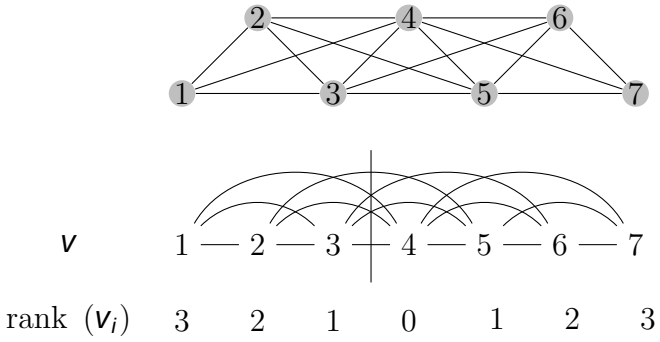
Proposition 4.6.8. *Proper interval k-trees have pathwidth k and are edge-maximal.*

Proof. Let G be a proper interval k-tree and let be a regular ordering of G.

First, observe that by taking X_i to be $\{\ (i);\ (i+k)\}$ for $1 \quad i < n \quad k$, $|X_i| = k+1$, and moreover, $|X_i \setminus X_{i+1}| = k$ for $1 \quad i < m$ For every $(\ (a);\ (b)) \in E(G)$ where $1 \quad a < b \quad n$, $b \quad a \quad k$, so $(\ (a);\ (b)) \in X_b$, i.e., every edge of G is in a bag. Finally, if $v \in V(G)$, then let ℓ be the index of its leftmost neighbour in . Then $v \in \{X_\ell, X_{\ell+1}, \ldots, X_{\ (v)}\}$ and no other bags, i.e., the bags containing v are consecutive in an ordering $X_1, \ldots, X_{n\ k}$.



(a) A proper interval 2-tree on seven vertices.



(b) A proper interval 3-tree on seven vertices.

Figure 4.6: Example proper interval k-trees, each illustrated two ways. The bottom illustration of the graph is a linear ordering which is also a regular ordering of the graph (and therefore also a cutwidth-minimal ordering). The vertical line through the linear ordering indicates a largest cut of the ordering.

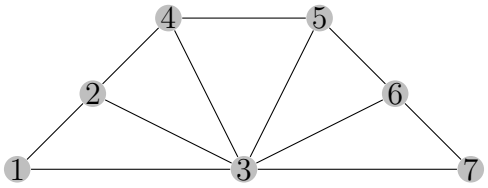


Figure 4.7: An edge-maximal 2-tree that is not a proper interval graph. The set $f1; 3; 6; 7g$ induces a claw, which means that the graph is not a proper interval graph by Theorem 4.6.6. On the other hand, G can be constructed by starting with $f1; 2; 3g$ and adding the vertices in the order indicated by the figure, showing that the graph is a 2-tree.

Therefore, we have a pathwidth decomposition which satisfies (1) and (2); since the biggest bag size is $k + 1$, $pw(G)$ k.

Second, let v 2 $V(G)$ n $(X_1$ [$X_{n\ k})$, if it exists. Since v 2 X_1, $(v) > k + 1$ as X_1 is defined to be the first $k + 1$ vertices of . Similarly, since v 2 $X_{n\ k}$, $(v) < n$ k as X_1 is defined to be the first $k + 1$ vertices of . Thus v has k neighbours to its left, and k neighbours to its right. Each of v's neighbours to its left, say f $(i); : : : ;$ $(i + (k$ $1))g$, first appears in X_i. Since v is also in $X_{(i+k)}$ as it has k neighbours to its right, v is in at least $k + 1$ bags.

Therefore, either no v 2 $V(G)n(X_1$ [$X_{n\ k})$ exists, or every such vertex is in at least $k+1$ bags; either way, we have satisfied condition (3). By Lemma 4.6.7, G is edge-maximal. □

However, it is important to note that proper interval k-trees are not the only edge-maximal graphs for a fixed pathwidth value. The graph in Figure 4.7 is also edge-maximal 2-tree but it is not a proper interval graph.

We now prove the main result for this section, which is a corollary of the next theorem and the following lemma.

Theorem 4.6.9 (Theorem 2.7, Yuan and Zhou [140]). *If G is a proper interval graph and is a regular ordering of G, then $cw( ) = cw(G)$.*

Lemma 4.6.10. *Let G be a proper interval that is also a k-tree. If is a regular ordering of G, then there exists an index i such that rank $((i))$ 0 for all 1 i j, rank $((i))$ 0 for all $j < i$ n, and $c(i) = cw( )$.*

Proof. Since $(v_1) = d_G(v_1)$ 0 and $(v_n) = d_G(v_n)$ 0, the fact that there is an index i such that rank $(\ (i))$ 0 for all 1 i j and rank $(\ (i))$ 0 for all $j < i$ n follows from the next claim.

Claim 4.6.11. *rank* (v_i) *rank* (v_{i+1}) *for all* 1 $i < n$.

Proof of claim: We break the proof into two cases, based on whether $i+1$ n $(k$ 1).

Case 1: $i+1 < n$ $(k$ 1). By construction, every vertex v_j has at most k neighbours to its right in , and exactly k if $j < n$ $(k$ 1). Therefore, succ $(v_i) =$ succ $(v_{i+1}) = k$. Also by construction, every vertex v_j has at most k neighbours to its left in . If pred $(v_i) = k$, then pred $(v_{i+1}) = k$ as well, otherwise, pred $(v_i) < k$ and pred $(v_{i+1}) =$ pred $(v_i) + 1$. Therefore, pred (v_{i+1}) pred (v_i).

Now observe that

$$\begin{aligned}\text{rank } (v_{i+1}) &= \text{succ } (v_{i+1}) \quad \text{pred } (v_{i+1}) \\ &= \text{succ } (v_i) \quad \text{pred } (v_{i+1}) \\ &\quad \text{succ } (v_i) \quad \text{pred } (v_i) \\ &= \text{rank } (v_i);\end{aligned}$$

as required.

Case 2: $i+1$ n $(k$ 1). By construction, every vertex v_j has at most k neighbours to its left in . Since $i+1$ n $(k$ 1), succ $(v_{i+1}) < k$, then succ $(v_i) =$ succ $(v_{i+1})+1$; therefore, succ (v_i) succ (v_{i+1}). Also by construction, every vertex v_j has at most k neighbours to its left in . If pred $(v_i) = k$, then pred $(v_{i+1}) = k$ as well, otherwise, pred $(v_i) < k$ and pred $(v_{i+1}) =$ pred $(v_i) + 1$. Therefore, pred (v_{i+1}) pred (v_i).

Now observe that

$$\begin{aligned}\text{rank } (v_{i+1}) &= \text{succ } (v_{i+1}) \quad \text{pred } (v_{i+1}) \\ &\quad \text{succ } (v_i) \quad \text{pred } (v_{i+1}) \\ &\quad \text{succ } (v_i) \quad \text{pred } (v_i) \\ &= \text{rank } (v_i);\end{aligned}$$

as required.

The only thing left to prove is that there is a maximum cut at index i. By Observation 3.2.7, for any ordering and 1 j n 1, c $(\ (j+1)) = \sum_{i=1}^{j+1}$ rank $(\ (i))$. Thus $cw(\ )$ is maximized at after v_i, as after this vertex, the total sum is non-increasing. □

Corollary 4.6.12. *If G is a proper interval graph and a k-tree, then $im(G) = 2cw(G)$.*

Proof. Let be a regular ordering of G, which exists by Theorem 4.6.10 and is cutwidth-minimal by Theorem 4.6.9. By Lemma 4.6.10, there is an index i of such that the ranks of all vertices before (i) are non-negative, and the ranks of all remaining vertices are all non-positive. By Lemma 3.4.2, $im(G) = 2cw(G)$. □

Proper Interval k-Trees and Bandwidth

We end this section with a conjecture relating the graphs in this section to another ordering problem, Bandwidth. We start by formally defining the problem.

Problem 4.6.13 (Bandwidth)**.** *Given a graph G and a positive integer k, determine if there is an ordering of G such that* $\max_{(u;v)2E(G)} fj \ (u) \quad (v)jg \quad k.$

Ideally, a solution to Bandwidth finds an ordering of G such that $W( ) \quad k$. As we will discuss in Section 5.1, Bandwidth is related to Optimal Linear Arrangement.

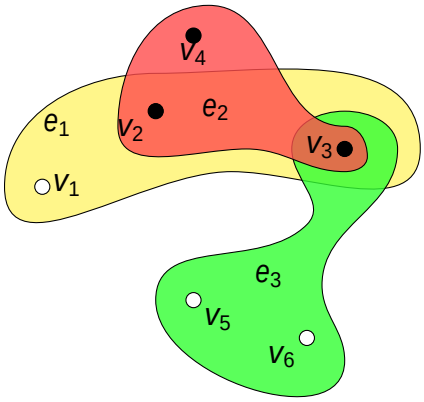
We conjecture that an $(n; k)$ proper interval k-tree is the edge-maximal graph with n vertices of bandwidth at most k. By "edge-maximal", we mean that adding any edge which is not present increases the bandwidth of the graph.

4.7 Vertex Cover Parameterizaton

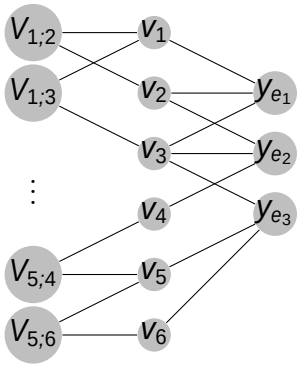
4.7.1 Kernelization Lower Bound

In this section show that there is likely no polynomial size kernel for Imbalance when the parameter is $vc(G)$, the size of a minimum vertex cover of the graph.

The proof uses an auxiliary problem, Hypergraph Minimum Bisection, which we define first. A *hypergraph* H is a pair $H = (V; E)$ where V is a set of vertices and $e \ 2 \ E$ is a subset of vertices in V (i.e., $e \quad V$ for all $e \ 2 \ E$). A subset $e \ 2 \ E$ is called a *hyperedge*. For a hypergraph $H = (V; E)$, the set V is sometimes called the *universe* of the hypergraph. A hypergraph is a *multihypergraph* if it allows multiple edges to exist which consist of the same subset of vertices in V. A *bisection* of V is a colouring $B : V \ ! \ f0; 1g$ such that $jB \ ^1(0)j = jB \ ^1(1)j = n=2$. For a hyperedge e, define the cost of e with respect to a bisection B as $cost(e; B) = \min(je \setminus B \ ^1(0)j; je \setminus B \ ^1(1)j)$. The cost of a bisection B is defined as $cost(B) = \sum_{e2E} cost(e; B)$.



(a) A hypergraph H with a bisection B of black- lled and white- lled vertices. Observe that $cost(B) = cost(e_1; B) + cost(e_2; B) + cost(e_3; B) = 1 + 0 + 1 = 2$.



(b) A graph G constructed from H. Each set $V_{a;b}$ is an independent set of size N where each vertex in the set is adjacent to v_a and v_b.

Figure 4.8: The reduction of Lemma 4.7.3. A (multi)hypergraph H is shown in (a) with $n = 6$ and $m = 3$, therefore $N = nm + 1 = 7$ in this example. Subfigure (b) shows the resulting graph G constructed from H.

Problem 4.7.1 (Hypergraph Minimum Bisection with parameter n)**.** *Given a multihypergraph H with n vertices, where n is even and an integer k, determine if there is a bisection of H with cost at most k?*

Hypergraph Minimum Bisection without a parameter is NP-complete (Cygan et al. [42]).

Cygan et al. [42] showed the following lemma for Cutwidth.

Lemma 4.7.2 (Lemma 12, Cygan et al. [42])**.** *There exists a polynomial-time algorithm that, given an instance of* Hypergraph Minimum Bisection *problem with n vertices, outputs an equivalent instance of the* Cutwidth *problem along with a vertex cover of size n for the graph of this instance.*

By adapting the proof of Lemma 4.7.2 and using Lemma 3.4.2 we can prove the following lemma.

Lemma 4.7.3. *There exists a polynomial-time algorithm that, given an instance of* Hypergraph Minimum Bisection *problem with n vertices, outputs an equivalent instance of the* Imbalance *problem along with a vertex cover of size n for the graph of this instance.*

Proof. Let $(H = (V; E); k)$ be an instance of Hypergraph Minimum Bisection. Then H is a hypergraph with n vertices and m edges; let $N = nm+1$. We will construct a graph $G = (V^0; E^0)$ such that $im(G)$ $2(n^2N=4+k)$ if and only if H has a bisection B such that $cost(B)$ k. The construction of G is exactly the same as the construction used for the proof of Lemma 4.7.2 in Cygan et al. [42]; we include it here for completeness.

We begin by taking the whole set V to be the set of vertices of G. For every distinct $u; v$ 2 V, we introduce N new vertices $x^i_{u;v}$ for $i = 1; 2; : : : ; N$, each connected only to u and v. Then, for every e 2 E we introduce a new vertex y_e connected to all v 2 e. Denote the set of all vertices $x^i_{u;v}$ by X and the set of all vertices y_e by Y. This concludes the construction. Observe that V is a vertex cover of G of size n. We now prove that H has a bisection with cost at most k if and only if G has cutwidth at most $n^2N=4 + k$.

Assume that H has a bisection B with cost at most k. Let us order the vertices of the graph G as follows. First, we order the vertices from V: we place B $^1(0)$ first, in any order, and then B $^1(1)$, in any order. Then, we place $x^i_{u;v}$ anywhere between u and v. At the end, for every e 2 E we place y_e at the beginning if at least half of the vertices of e are in B $^1(0)$, and in the end otherwise. Vertices y_e at the beginning and at the end are arranged in any order.

Now we prove that the cutwidth of the constructed ordering is at most $n^2N=4+k$. Consider any cut C, dividing the order on $V(G)$ into a first part V_1 and a second part V_2. Suppose that $jV_1 \setminus V j = n=2$ ` for some $n=2$ ` $n=2$, thus $jV_2 \setminus V j = n=2 + `$. Observe that $C$ cuts exactly $N(n=2$ `$)(n=2 + `) = n^2N=4$ `$^2N$ edges between V and X. Note that there are not more than $nm < N$ edges between V and Y. Therefore, if ` 6= 0, then C can cut at most $n^2N=4$ $N + nm < n^2N=4 + k$ edges.

We are left with the case when ` = 0. Observe that $V_1 \setminus V = B$ $^1(0)$ and $V_2 \setminus V = B$ $^1(1)$. Moreover, the cut C cuts exactly $n^N=4$ edges between sets V and X. As far as edges between V and Y are concerned, for every hyperedge e 2 E, the cut C cuts exactly $cost(e; B)$ edges incident on y_e. As $cost(e; B)$ k, the cut C cuts at most $n^2N=4 + k$ edges.

Now assume that there is an ordering of vertices of G that has cutwidth at most $n^2N=4 + k$. We construct a bisection B of H as follows. Let $B(v) = 0$ for

every v among the first $n/2$ vertices from V with respect to the ordering, and $B(v) = 1$ for v among the second $n/2$ vertices. We now prove that the cost of this bisection is at most k.

Let C be any cut dividing the order into the first part V_1 and the second part V_2, such that $V_1 \setminus V = B^{-1}(0)$ and $V_2 \setminus V = B^{-1}(1)$. As the cutwidth of the ordering is at most $n^2N/4 + k$, C cuts at most $n^2N/4 + k$ edges. Observe that C needs to cut at least $n^2N/4$ edges between sets V and X, therefore it cuts at most k edges between sets V and Y. For every hyperedge $e \in E$, C cuts at least $cost(e; B)$ edges incident to y_e, thus $cost(B) \leq k$.

Therefore, if $cost(B) \leq k$, there is an ordering π such that $cw(\pi) = n^2N/4 + k$, a largest cut always exists after $n/2$ vertices of G, and that $\chi(x^i_{u;v}) = 0$ for any $u; v \in V$ and $1 \leq i \leq N$.

We first prove the following claim:

Claim 4.7.4. *$im(G) \leq 2(n^2N/4 + k)$ if and only if $cost(B) \leq k$ for some bisection B.*

Proof of claim: First, suppose that $cost(B) \leq k$ for some bisection B. By the proof above, there is an ordering π of G such that $cw(\pi) = n^2N/4 + k$. Let $Y_\ell \ldots Y$ be the vertices appearing before any vertex of V, and $Y_r = Y \cap Y_\ell$ be those appearing at the end of the ordering.

If $\chi(v) = 0$ for any $v \in V(G)$, we are done; we may therefore assume $\chi(v) \neq 0$. In particular, this means we only need to deal with vertices of V and Y, as those vertices $x^i_{u;v}$ for any $u; v$ and $1 \leq i \leq N$ have imbalance zero. Recall that rank $_\pi(v) = |N(v) \cap \ldots {}_v|$ $|N(v) \setminus \ldots {}_v| = \text{succ}_\pi(v) - \text{pred}_\pi(v)$.

Suppose $y_e \in Y_\ell$. Since Y is an independent set, all neighbours of y_e are to the right of y_e in π. That is, rank $_\pi(y_e) = \text{succ}_\pi(y_e) - \text{pred}_\pi(y_e) = \text{succ}_\pi(y_e) \geq 0$. Similarly, for $y_2 Y_r$, since Y is an independent set, all neighbours of y_e are to the left of y_e in π. That is, rank $_\pi(y_e) = \text{succ}_\pi(y_e) - \text{pred}_\pi(y_e) = - \text{pred}_\pi(y_e) \leq 0$.

Suppose $v \in V(G)$ is within the first $n/2$ vertices of $V(G)$ in π; we claim that rank $_\pi(v) \geq 0$.

Let v be the ith vertex of V in π for $1 \leq i \leq n/2$. We will show that rank $_\pi(v) \geq 0$. Then pred $_\pi(v) = |N(v) \setminus Y_\ell| + N(i - 1)$ as v is adjacent to any neighbours in Y_ℓ, and for each $v' \in V$ such that $v' < v$ (of which there are $i - 1$), each vertex $x^j_{v;v'}$ is between v' and v in π (of which there are N for each such v'). Similarly, succ $_\pi(v) = |N(v) \setminus Y_r| + N(n - i)$, as v is adjacent to any neighbours in Y_r, and for each $v' \in V$ such that $v < v'$ (of which

there are $n \quad i$), each vertex $x^j_{v;v'}$ is between v and v' in $\quad$ (of which there are N for each such v').

Since $n{=}2 \quad i$, $(n \quad i) \quad (i \quad 1) > 0$; let $d = (n \quad i) \quad (n \quad 1) > 0$. We have

$$\begin{aligned}
\text{rank}\ (v) &= \text{succ}\ (v) \quad \text{pred}\ (v) \\
&= jN(v) \setminus Y_r j + N \quad (n \quad i) \quad jN(v) \setminus Y_\cdot j \quad N \quad (i \quad 1) \\
&= jN(v) \setminus Y_r j \quad jN(v) \setminus Y_\cdot j + N \quad ((n \quad i) \quad (i \quad 1)) \\
&= jN(v) \setminus Y_r j \quad jN(v) \setminus Y_\cdot j + N \quad d:
\end{aligned}$$

Since $jN(v) \setminus Y_r j \quad 0$ and

$$jN(v) \setminus Y_\cdot j \quad m \quad nm + 1 = N \quad Nd;$$

we have that $jN(v) \setminus Y_r j \quad jN(v) \setminus Y_\cdot j + N \quad d \quad 0$ and therefore $\text{rank}\ (v) \quad 0$.

Similarly, if $v \ 2 \ V(G)$ is within the last $n{=}2$ vertices of $V(G)$ in $\quad$, we will show that $\text{rank}\ (v) \quad 0$. Then $\text{pred}\ (v) = jN(v) \setminus Y_\cdot j + N \quad (i \quad 1)$ as v is adjacent to any neighbours in $Y_\cdot$, and for each $v' \ 2 \ V$ such that $v' < \ v$ (of which there are $i \quad 1$), each vertex $x^j_{v;v'}$ is between v' and v in $\quad$ (of which there are N for each such v'). Similarly, $\text{succ}\ (v) = jN(v) \setminus Y_r j + N \quad (n \quad i)$, as v is adjacent to any neighbours in Y_r, and for each $v' \ 2 \ V$ such that $v < \ v'$ (of which there are $n \quad i$), each vertex $x^j_{v;v'}$ is between v and v' in $\quad$ (of which there are N for each such v').

Since $n{=}2 > i$, $(n \quad i) \quad (i \quad 1) < 0$; let $d = j(n \quad i) \quad (n \quad 1)j$. We have

$$\begin{aligned}
\text{rank}\ (v) &= \text{succ}\ (v) \quad \text{pred}\ (v) \\
&= jN(v) \setminus Y_r j + N \quad (n \quad i) \quad jN(v) \setminus Y_\cdot j \quad N \quad (i \quad 1) \\
&= jN(v) \setminus Y_r j \quad jN(v) \setminus Y_\cdot j + N \quad ((n \quad i) \quad (i \quad 1)) \\
&= jN(v) \setminus Y_r j \quad jN(v) \setminus Y_\cdot j \quad N \quad d:
\end{aligned}$$

Since $jN(v) \setminus Y_\cdot j \quad 0$ and

$$jN(v) \setminus Y_r j \quad m \quad nm + 1 = N \quad Nd;$$

we have that $jN(v) \setminus Y_r j \quad jN(v) \setminus Y_\cdot j \quad N \quad d \quad 0$ and therefore $\text{rank}\ (v) \quad 0$.

Therefore, by Lemma 3.4.2, $im(G) = 2cw(G) = 2(n^2N{=}4 + k)$.

Suppose instead that $im(G) \quad 2(n^2N{=}4+k)$, which implies that $im(G){=}2 \quad n^2N{=}4+k$. By Lemma 3.1.1, $cw(G) \quad im(G){=}2$ for any graph G. Therefore, we have $cw(G) \quad im(G){=}2 \quad n^2N{=}4 + k$ and by the proof above, $cost(B) \quad k$. This completes the proof of the claim.

The lemma is immediate from the claim. □

Now we aim to prove the main result of this subsection, Theorem 4.7.5, which is stated next.

Theorem 4.7.5. Imbalance *parameterized by the size of the vertex cover does not admit a polynomial kernel, unless* NP coNP=poly.

We require the following lemma, also due to Cygan et al. [42].

Lemma 4.7.6 (Lemma 9, Cygan et al. [42])**.** Hypergraph Minimum Bisection, *parameterized by the the size of the universe, does not admit a polynomial kernel, unless* NP coNP=poly.

Using Lemma 4.7.6 and Theorem 4.7.8 below (which requires Definition 4.7.7, also below), Cygan et al. [42] were able to show an analogous result of Theorem 4.7.5 for Cutwidth. Note that the Definition 4.7.7 is similar to that of an FPT reduction, however, we require that the new parameter value is computed via a polynomial function.

Definition 4.7.7 (Definition 7, Bodlaender et al. [13])**.** *Let P and Q be parameterized problems. We say that P is polynomial parameter reducible to Q if there exists a polynomial time computable function $f : \Sigma$ N $! \Sigma$ N, and a polynomial p, such that for all $(x;k)$ 2 Σ N the following holds: $(x;k)$ 2 P if and only if $(x';k') = f(x;k)$ 2 Q and k' $p(k)$.*

Theorem 4.7.8 (Theorem 8, Bodlaender et al. [13])**.** *Let P and Q be parameterized problems and P' and Q' be the unparameterized versions of P and Q respectively. Suppose that P' is* NP*-hard and Q' is in* NP*. Assume there is a polynomial parameter transformation from P to Q. Then if Q admits a polynomial kernel, so does P.*

Finally, we can prove Theorem 4.7.5.

Proof of Theorem 4.7.5. Use Lemma 4.7.3, Lemma 4.7.6 and Theorem 4.7.8; the result is immediate as Imbalance and Hypergraph Minimum Bisection are both NP-complete. □

4.7.2 Improved Imbalance FPT Algorithm for Graphs of Bounded Vertex Cover Number

In this section, we note that the approach of Cygan et al. [42] that was used to solve Cutwidth is applicable to Imbalance. Therefore, we will show that given a graph

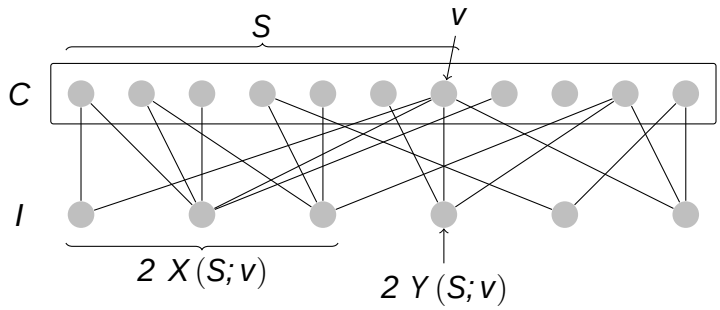


Figure 4.9: Illustration of the sets $X(S;v)$ and $Y(S;v)$, redrawn from Cygan et al. [42].

$G = (C \uplus I; E)$ with a vertex cover C of size k, we can compute the imbalance of G in time $O(2^k n^{O(1)})$. The solution uses dynamic programming and is an improvement over the previous algorithm by Fellows et al. [52], who obtained an $O(2^{2^{O(k)}} n^{O(1)})$ algorithm. Fellows et al. were not concerned with optimizing the running time of their algorithm, only showing that an FPT algorithm exists for this problem and parameter.

Explicitly stated, the main result of this section is the following theorem. We base its proof off of the proof of the analogous theorem for Cutwidth by Cygan et al. [42].

Theorem 4.7.9. *Let G be a graph with vertex cover of size k. There is an algorithm to solve* Imbalance *in time $O(2^k n^{O(1)})$. Therefore, there is a $O(2^{n/2} n^{O(1)})$-time algorithm for* Imbalance *on bipartite graphs.*

Let $G = (C \uplus I; E)$ for a vertex cover C and an independent set I. The proof uses the notion of *C-good orderings* and *C-good pre x orderings*. A C-good ordering is a refinement of the orderings provided by Theorem 4.4.1. In addition to requiring that every vertex of $v \in I$ has $(v) \in \{0,1\}$ depending on the parity of $|N_G(v)|$, a C-good ordering requires that the vertices in I between any two consecutive vertices of C in are sorted by rank (v) in ascending order. Thus, between any two consecutive vertices c_i, c_{i+1} in C, $(\ _{>c_i} \setminus\ _{<c_{i+1}}) \cap I$ has all vertices with non-negative rank after those with negative rank. The existence of such an ordering follows from Theorem 4.4.1 and the fact that the imbalance of a vertex is not affected by swapping it past a non-neighbour.

Using such orderings, we can now prove Theorem 4.7.9.

Proof of Theorem 4.7.9. Let $G = (C \uplus I; E)$ for a vertex cover C and an independent set I. We will establish a recurrence relation that can be used to compute $im(G)$, but need to define several sets which are used in the formulation first.

Partition I into sets I^{odd}, which contains only odd-degree vertices, and I^{even}, which only contains even-degree vertices.

We define two sets $X(S;v)$ and $Y(S;v)$ for $S \quad C$ and $v \ 2 \ S$ (see Figure 4.9). Define $X(S;v)$ as $X(S;v) = X^{even}(S;v) \ [\ X^{odd}(S;v)$, where

- $X^{odd}(S;v) = fu \ 2 \ I^{odd} \quad jN(u) \ \setminus \ (S \ n \ fvg)j > jN(u) \ n \ (S \ n \ fvg)jg$, and
- $X^{even}(S;v) = fu \ 2 \ I^{even} \quad jN(u) \ \setminus \ Sj > jN(u) \ n \ Sjg$.

Finally, define $Y(S;v) = fu \ 2 \ N(v) \quad j(N(u) \ n \ fvg) \ \setminus \ Sj = j(N(u) \ n \ fvg) \ n \ Sjg$. clearly $Y(S;v) \quad I^{odd}$. Observe that $X(S;v) \ \setminus \ Y(S;v) = ;$ for any $S \quad C$ and $v \ 2 \ S$.

If $\quad$ is a C-good ordering, let $\quad_C = fc_1;:::;c_kg$.

Observation 4.7.10 (Observation 3, Cygan et al. [42]). *In a C-good ordering $\quad$, let i be an integer such that $c_i \ 2 \ C$ and $S = \quad_i \setminus C$. Then $X(S;c_i) \quad \quad_i \setminus I \quad X(S;c_i) \ [\ Y(S;c_i)$.*

We now define *C-good pre x orderings* as in Cygan et al. [42]. For an ordering $\quad$, let $C_{i \ 1}$ be the first $i \quad 1$ vertices of C.

> A prefix ordering is an ordering $\quad$ of a set of vertices $V^0 \quad V = C \ [\ I$. A prefix-ordering such that $\quad(j \ j) = c_t$ for some $c_t \ 2 \ C$ is C-good if the following conditions are satisfied:
>
> 1. For every vertex $v \ 2 \quad \setminus I$ of even degree, rank $(v) = 0$ and for every vertex $v \ 2 \quad \setminus I$ of odd degree, rank $(v) \ 2 \ f \quad 1;1g$.
> 2. $X(\quad \setminus C; \quad (i)) \quad \setminus I \quad X(\quad \setminus C; \quad (i)) \ [\ Y(\quad \setminus C; \quad (i))$.
> 3. For every vertex $v \ 2 \ X(\quad \setminus C;c_i)$ such that rank $(v) \quad 0$ and $c_i \ 2 \quad \setminus C$ we have $c_i < \quad v$ if and only if $jN(v) \ \setminus \ C_{i \ 1}j \quad jN(v) \ n \ C_{i \ 1}j$.
> 4. For every vertex $v \ 2 \ X(\quad \setminus C;c_i)$ such that rank $(v) < 0$ and $c_i \ 2 \quad \setminus C$ we have $c_i < \quad v$ if and only if $jN(v) \ \setminus \ C_{i \ 1}j < jN(v) \ n \ C_{i \ 1}j$.
> 5. Between any two consecutive $(c_i;c_{i+1}) \ 2 \ C \setminus \ $, the vertices with rank $(v) <$ 0 come before all vertices with rank $(v) \quad 0$.
>
> Comparing the properties of C-good orderings and C-good prefix orderings it is easy to see that the following lemma holds.

Lemma 4.7.11 (Lemma 4, Cygan et al. [42]). *Let $\quad = v_1:::v_n$ be a C-good ordering and let $\quad$ be the restriction of $\quad$ to the $\quad$rst t vertices such that $\quad(t) \ 2 \ C$. Then $\quad$ is a C-good pre x ordering.*

Finally, let $L_i(v) = N(v) \setminus (X(S;v) \cup S \cup Y_i(S;v))$ and $R_i(v) = N(v) \cap L_i(v)$, where $Y_i(S;v)$ is an arbitrary subset of $Y(S;v)$ with size i.

We are now ready to establish the following recurrence relation, using the sets X and Y as defined above. Let $T(S;v)$ be the minimum value of $im(\ )$ where the minimum is taken over all C-good prefix orderings with $\setminus C = S$ and v being the last vertex of .

Claim 4.7.12. *$T(S;v) = \min_{u \in S} \min_{0 \le i \le |Y(S;v)|} T(S \cap fvg;u) + |Y(S;v)| + |L_i(v)| \quad |R_i(v)|$ for $S \quad C$ and $v \in S$.*

Proof of claim: The recurrence relation computes the imbalance of "extending" the previous best prefix ordering with a new vertex of C, and adding the imbalances for other vertices which should also be counted by such an extension.

$T(S \cap fvg;u)$ is the minimum imbalance from the first $|S| \quad 1$ vertices and vertices to the left of the last vertex in the corresponding prefix that satisfies $T(S \cap fvg;u)$.

For any set $S \quad C$ and vertex $v \in S$, every vertex $u \in Y(S;v)$ has odd degree and is adjacent to v and satisfies $|(N(u) \cap fvg) \setminus S| = |(N(u) \cap fvg) \cap S|$. Therefore, every vertex in $Y(S;v)$ will have imbalance 1, and this set contributes $|Y(S;v)|$ to the total imbalance.

We now consider the vertices of $X(S;v)$. Since vertices in $X(S;v)$ which have even degree do not increase the imbalance (as they are placed so that they have imbalance 0), the extension of a C-good prefix by vertices in $X(S;v)$ matters only if they have odd degree. Now consider the odd degree vertices of $X(S;v)$ and observe that $Y(S \cap fvg;u) \quad X(S;v)$; in fact, $Y(S \cap fvg;u) \quad X^{odd}(S;v)$. In particular, this means that we have added the imbalance of $x \in X^{odd}(S;v)$ as part of the contribution of $Y(S \cap fvg;u)$, so it is not necessary to count it again.

Therefore, only the imbalance of v remains to be counted, and it should be minimal. To minimize the imbalance of v, for every possible pair $(\ ; \ ) (\ ; \quad 0)$ such that $+ =$ $|Y(S;v)|$ we place vertices of $Y(S;v)$ on the left of v and the remaining vertices of $Y(S;v)$ on the right v. The imbalance of v for a given partition of $Y(S;v)$ is $|L_i(v)|$ $|R_i(v)|$. We choose the partition of $Y(S;v)$ that minimizes this value.

Finally, note that $im(G) = \min_{v \in C} T(C;v)$.

The recurrence relation implies a dynamic programming algorithm. In the base case—for small sized subsets S—we try all possible permutations of S and fill in the gaps as appropriate. In larger cases, we append a vertex of the vertex cover to a previous instance according to the recurrence relation in Claim 4.7.12. We now analyse the time complexity of such an algorithm.

- If $|S| = 1$, let $\{v\} = S$. The value $T(S; v)$ is the imbalance of the optimal ordering $= L \{v\}$ for some set $L \subseteq I$, since the C-good prefix ordering must end with a vertex in C. Any vertices $w \in I$ such that $N(w) = \{v\}$ must be placed in either L or later in the ordering; choose $|L|$ such that assuming all non-pendant neighbours of v are to its right, the imbalance of v is minimal. Determining and computing $T(S; v)$ requires at most linear time.

- If $|S| = 2$, let $\{u\} = S \setminus \{v\}$. The value $T(S; v)$ is the imbalance of the optimal ordering $= L \{u\} M \{v\}$ for some sets $L, M \subseteq I$ since the C-good prefix ordering must end with a vertex in C. Any vertices $w \in I$ such that $N(w) = \{u, v\}$ must be placed in M. Any vertices $w \in I$ such that $N(w) = \{u\}$ must be placed in either L or M, split them among these sets so that assuming all non-pendant neighbours of u are to its right, the imbalance of u is minimal. Some vertices $w \in I$ such that $d(w) = 3$ and $u, v \in N(w)$ may require placement in M: choose the appropriate amount so that the imbalance of v is minimized given that the vertices already placed to its left are fixed. Determining and computing $T(S; v)$ takes at most linear time.

- Otherwise, $|S| > 2$, and $T(S; v) = \min_{u \in S} \min_{0 \le i \le |Y(S;v)|} T(S \setminus \{v\}; u) + |Y(S; v)| + |L_i(v)| - |R_i(v)|$. Determining and computing $T(S; v)$ requires quadratic time at worst for each $u \in S$.

Therefore, for a vertex cover of size $|C|$, we consider $|C| - 2$ subsets each of size $|C| - 1$ such that each subset does not have the $|C|$th element in them. There are at most $O(|C| \cdot 2^{|C|})$ subproblems, and each one takes at most quadratic time to solve. The total running time is therefore $O(|C|^3 \cdot 2^{|C|})$. □

4.8 Remarks on Previous Attempts

In this section, we make some remarks on previously claimed results for Imbalance which are not correct.

4.8.1 FPT Algorithm for Twin Cover Number

Previously, we claimed (Gorzny and Buss [68]) that both Imbalance and Cutwidth are FPT when parameterized by the twin cover number of a graph. However, Misra and Mittal [111] showed that our approach was not correct; we explain the error in this section.

Recall that a *twin cover* of a graph G is a set $T \subseteq V$ such that for every edge $(u, v) \in E$, either $\{u, v\} \cap T \neq \emptyset$ or u and v are true twins (Ganian [57]). Every vertex cover is a twin cover, though not every twin cover is a vertex cover. The minimum size of a twin cover for a graph is the graphs *twin cover number* and is denoted $tc(G)$.

The approach that was attempted for graphs with bounded twin cover number is similar to the one used for restricted twin cover number ($rtc(G)$) for Cutwidth (Section 3.7). This approach may also be applicable for Imbalance for graphs of bounded restricted twin cover number, but since the neighbourhood diversity of a graph generalizes restricted twin cover number (see Section 2.5.2), the bounded neighbourhood diversity result for Imbalance already implies that the problem is FPT when the parameter is $rtc(G)$.

The approach taken by Gorzny and Buss [68], oversimplifying, was the following:

1. Obtain a minimum twin cover T of G;
2. Contract the edges which do not have an endpoint in T and call the resulting graph $G^{\emptyset}$;
3. Observe that T is now a vertex cover of $G^{\emptyset}$ (Lemmas 2.5.1 and 2.5.2);
4. Apply the ILP for Imbalance on graphs of bounded vertex cover number of Fellows et al. [52], which uses Theorem 3.7.2, for every permutation of the vertices of T;
5. Return the minimum ordering found;
6. Replace the contracted vertices in $G^{\emptyset}$ with the un-contracted sets in G.

The result was claimed to be imbalance-minimal. The approach is attractive: it is simple and tries to exploit the useful connection of Lemmas 2.5.1 and 2.5.2. However, there is a simple flaw: the ILP cannot differentiate between contracted sets of vertices of different sizes.

To illustrate this, consider the superfragile graph in Figure 4.10. The graph G has a vertex cover $T = \{1\}$. The three disjoint cliques $\{2, 3\}$, $\{4\}$, $\{5, 6, 7, 8\}$ are contracted into vertices a, b, and c to obtain $G^{\emptyset}$. It is easy to verify that T is a vertex cover of $G^{\emptyset}$. There is only one permutation of T, so only one ILP would be constructed. Necessarily, any minimum ordering found would place at least one of $\{a, b, c\}$ to the left of 1 in the ordering, and the remaining vertices of $\{a, b, c\}$ to the right of 1 (or vice-versa). Indeed, for this example, any such ordering that splits $\{a, b, c\}$ as evenly as possible is minimum for $G^{\emptyset}$. However, since the ILP treats those vertices equally, it does not know that c should

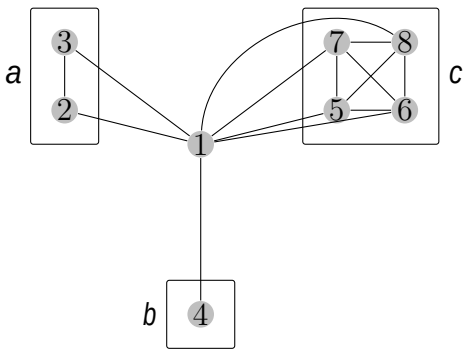


Figure 4.10: A superfragile graph G illustrating how the FPT algorithm for Imbalance fails for $tc(G)$. The set $T = \{1\}$ is a twin cover of G. Each of the disjoint cliques $K_2 = \{3, 2\}$, $K_1 = \{4\}$, and $K_4 = \{5, 6, 7, 8\}$ are contracted into vertices a, b, and c respectively in G'.

not be on the same side of 1 as a or b. Thus, if the result was $= \{\{a\}\ \{c\}\ \{1\}\ \{b\}\}$, then $im(\ ) = im(G')$. However, replacing $\{a, b, c, g\}$ by the sets of vertices that were contracted to obtain the vertices, we obtain $' = \{\{3, 2\}\ \{5, 6, 7, 8\}\ \{1\}\ \{4\}\}$ which has a greater imbalance than, say, $\{\{5, 6, 7, 8\}\ \{1\}\ \{3, 2\}\ \{4\}\}$, and is therefore not imbalance-minimal.

Misra and Mittal [111] overcame this issue by also requiring the *size* of a largest clique outside of the twin cover is also a part of the parameter. This issue was corrected for Cutwidth in Section 3.7 by using the restricted twin cover number, which was defined exactly to overcome this issue. The restricted twin cover overcomes the issue by ensuring that the *number* of (non-singleton) cliques outside the twin cover is also a part of the parameter.

4.8.2 Proper Interval (Bipartite) Graphs

We also attempted to show that Imbalance has linear time solutions for proper interval graphs and proper interval bipartite graphs, a.k.a. bipartite permutation graphs (Gorzny and Buss [68], Gorzny [67]). In both cases, flaws exist in the proofs, though we conjecture that the theorem statements are true (see Chapter 7).

The method for both of these classes of graphs are the same, though the details differ. The proofs were set up in order to apply induction on either the number of maximal cliques (proper interval graphs) or the number of vertices (bipartite permutation graphs). Then, after establishing some suitable base cases, the inductive step would be as follows:

- Determine one "end" of the graph G, with respect to its linear structure. In the case of proper interval graphs, this would be a simplicial vertex; such a vertex can be the rightmost or leftmost end of a regular ordering (see Section 4.6) of the graph. For bipartite permutation graphs, it would be one of the end points of the strong ordering (see Section 2.3.3) of the graph.
- Remove that end of the graph G, and apply the induction hypothesis to the resulting graph G_1.
- Identify another "end" of the graph G, with respect to its linear structure. For example, in the proper interval graph setting, this would be a simplicial vertex of the graph that is farthest from the simplicial vertex chosen in step 1.
- Remove this end of the graph, and apply the induction hypothesis to the resulting graph G_2.
- Identify a vertex v common to both G_1 and G_2 that has its closed neighbourhood in the same ordering in both G_1 and G_2, and then take an imbalance-minimal ordering of G_1 up to that vertex, and then use G_2 from that point onward.

The difficulty in the inductive step arises from the last step described above. Showing that such a vertex always exists in both graphs, along with its closed neighbourhood, is not trivial. However, as long as the inductive step deals with sufficiently large graphs, this appears possible. The "gluing" operation of the last step was also fairly straightforward, provided the structure of the optimal orderings was well-defined.

As an example, for the case of proper interval graphs, the conjecture is that a regular ordering was imbalance-minimal. If one can show that there is a vertex v whose closed neighbourhood has the same relative ordering in imbalance-minimal orderings of G_1 and G_2, then the gluing is technique a good idea. First, the resulting ordering is easily shown to be a regular ordering of the graph G, as both the imbalance-minimal orderings of G_1 and G_2 are too. Second, the optimality of the ordering arises from the fact that every vertex before v might as well be in an ordering of G_1 rather than G, since its closed neighbourhood would have also been contained in G_1. Similarly, any vertex that came after v might as well be in an ordering of G_2 rather than G. That is, from the point of view of a vertex only in G_1, it does not care about vertices to the right of v, i.e., those vertices only in G_2, and vice-versa. Then the concern is vertices which may be present in both graphs, but this can be handled by choosing v carefully, and relying on the properties of regular orderings. Thus, if G_1 and G_2 were optimal, so was the resulting ordering. This is illustrated in Figure 4.11.

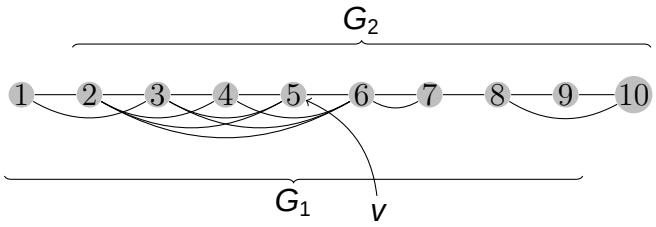


Figure 4.11: A proper interval graph G illustrating how to use induction for Imbalance. The vertices 1 and 10 can be considered the "ends" of the graph. The ordering is in fact imbalance-minimal, as is $v(G_1)]$ and $v(G_2)$, and all of these orderings are regular orderings. The vertices to the left of v (and v itself) have the same imbalance in $v(G_1)$ and , while the vertices to the right of v (and v itself again) have the same imbalance in $v(G_2)$ and . These vertices are indifferent to what happens at the other "end" of the graph. Putting $v(G_1)$ and $v(G_2)$ together by using the ordering $G_1 \cap G_2$ $G_1 \setminus G_2$ $G_2 \cap G_1$ keeps those imbalances, and results in the ordering .

It turns out that the real issue is the base cases for these proofs.

For proper interval graphs, the induction used maximal cliques (Gorzny and Buss [68]). In this case, one would not only remove the leftmost or rightmost simplicial vertex, but the maximal clique containing it. Then, instead of finding a vertex v to use as a gluing point in the inductive case, one would consider a maximal clique. There is nothing substantially different about this than the method described above; it simplifies book keeping in some ways since there are fewer things to count. However, it requires finding a maximal clique in the inductive step which is not adjacent to either end of a regular ordering. The base case was incorrectly assumed to have only a small finite number of maximal cliques in such a scenario (3 or 5), which was not true. As a result, this special "common" clique is not always found.

The proof would be repaired if the base case can be repaired. This boils down to showing that all proper interval graphs with a non-empty set of universal vertices have an imbalance-minimal ordering which is also a regular ordering. This is because if the graph has a non-empty set of universal vertices, any maximal clique in the proper interval graph contains at least one of the endpoints of the regular ordering. Since these graphs have a non-empty universal set of vertices, they also have a diameter of at most two. This turns out to be difficult – one can find diameter 2 proper interval graphs which do not satisfy the requirements of Lemma 3.4.2 (see Figure 4.12), which removes the most straightforward

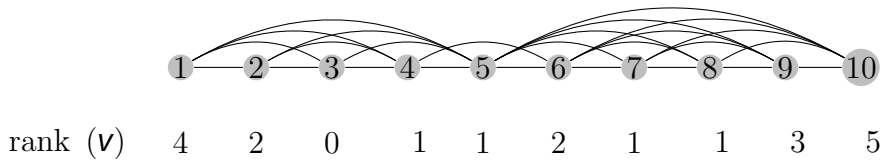


Figure 4.12: A proper interval graph with diameter 2 which does not satisfy the requirements of Lemma 3.4.2. Note that the ordering is a regular ordering, and that it is imbalance-minimal.

option. Worse, there is a lack of helpful lemmas. The most helpful would be a lemma saying that a simplicial vertex of a proper interval graph appears first in some imbalance-minimal ordering of the graph, but the proof of Corollary 4.4.3 does not apply since there is no guarantee that the open neighbourhood is consecutive in some imbalance-minimal ordering. This also means we cannot adapt the proof of Lemma 4.4.4.

For bipartite permutation graphs, the same is also more or less true. The conjecture is that every bipartite permutation graph has an imbalance-minimal ordering which agrees with a strong order of the graph (which always exists by Theorem 2.3.5). The induction by Gorzny [67] was on the number of vertices, rather than the number of maximal cliques, but follows the same outline, and suffers again from a base case which is not clear. The base case is more difficult to explain, as there were more cases to consider. Instead of diameter exactly two graphs being problematic, the issue is graphs with diameter at most three. We will explain a bit more, though perhaps it is easier to see this by simply noting that bipartite graphs of diameter at most 2 are complete bipartite graphs, for which the problem is solved by Lemma 4.3.6. To explain, suppose that a bipartite permutation graph has a bipartition $(A; B)$. The "ends" of the graph are less clear: each partite set has an "end", that is, there is a leftmost vertex in A and a leftmost vertex in B, and similarly there are two "rightmost" vertices. Similar to the issue for proper interval graphs, complications arise when ends are at distance two. In particular, even if $d_G(a; a') = 2$ for any vertices $a; a' \in A$, which would imply a vertex $b \in B$ which is adjacent to all of A, there may be pendant vertices $b' \in B$ adjacent to only the left- or rightmost vertex in A according to a strong ordering. In such a case, the diameter is at least three. So the (corrected) proof for bipartite permutation graphs requires an even more complicated base case.

We make one final remark on imbalance-minimal orderings of proper interval graphs. We conjecture that regular orderings are imbalance-minimal, and we do so in part because they also fit the description of an optimal ordering of the graph that exists by Theorem 4.4.2: regular orderings place each class of true twins consecutively in the ordering. This

is shown by the next lemma.

Lemma 4.8.1. *If is a regular ordering of some proper interval graph G, then each equivalence class of true twins C of G is consecutive in .*

Proof. Let $G = (V; E)$ and suppose that is a regular ordering where $a <$ $x <$ b and $N[a] = N[b]$. We will show that $N[x] = N[a] = N[b]$, which will show that between any two true twins in a regular ordering, there can only be more true twins from the same equivalence class. Since a and b are true twins, $(a; b) \in E$.

First, since $(a; b) \in E$ and is a regular ordering, we must have that $(a; x) \in E$ and $(x; b) \in E$, as $V(a; b)$ must be a clique by definition of a regular ordering.

We now prove that $N(a) \cap \{b; x\} = N(x) \cap \{a; b\}$ in two claims.

Claim 4.8.2. *If $a' \in N(a) \cap \{b; x\}$, then $a' \in N(x) \cap \{a; b\}$.*

Proof of claim: Since $a' \in N(a)$, $(a; a') \in E$. We proceed by cases based on where a' is in , relative to a, b, and x.

Case 1: $a <$ $x <$ $b <$ a'. The existence of the edge $(a; a')$ implies that $(x; a') \in E$ as $V(a; a')$ must be a clique by definition of a regular ordering.

Case 2: $a <$ $x <$ $a' <$ b. The existence of the edge $(a; b)$ implies that $(x; a') \in E$ as $V(a; b)$ must be a clique by definition of a regular ordering.

Case 3: $a <$ $a' <$ $x <$ b. The existence of the edge $(a; b)$ implies that $(x; a') \in E$ as $V(a; b)$ must be a clique by definition of a regular ordering.

Case 4: $a' <$ $a <$ $x <$ b. Since a and b are true twins, if $(a; a') \in E$, then $(b; a') \in E$. The existence of the edge $(b; a')$ implies that $(x; a') \in E$ as $V(a'; b)$ must be a clique by definition of a regular ordering.

Thus, $(a'; x) \in E$ in every case, and the claim is proved.

Claim 4.8.3. *If $x' \in N(x) \cap \{a; b\}$, then $x' \in N(a) \cap \{b; x\}$.*

Proof of claim: Since $x' \in N(x)$, $(x; x') \in E$. We proceed by cases based on where x' is in , relative to a, b and x.

Case 1: $a <$ $x <$ $b <$ x'. The existence of the edge $(x; x')$ implies that $(x'; b) \in E$ as $V(x; x')$ must be a clique by definition of a regular ordering. Since a and b are true twins, $(x'; a) \in E$ too.

Case 2: $a < x < x' < b$. The existence of the edge $(a; b)$ implies that $(x'; b) \in E$ and $(x'; a) \in E$ as $V(a; b)$ must be a clique by definition of a regular ordering.

Case 3: $a < x' < x < b$. The existence of the edge $(a; b)$ implies that $(x'; b) \in E$ and $(x'; a) \in E$ as $V(a; b)$ must be a clique by definition of a regular ordering.

Case 4: $x' < a < x < b$. The existence of the edge $(x; x')$ implies that $(x'; a) \in E$ as $V(x'; x)$ must be a clique by definition of a regular ordering. Since a and b are true twins, $(x'; b) \in E$ too.

Thus, $(x'; a) \in E$ and $(x'; b) \in E$ in every case, and the claim is proved.

Combining the previous two claims, we have that $N(a) \setminus \{b\}; x\} = N(x) \setminus \{a; b\}$. Therefore

$$N[a] = N(a) \setminus \{b; x\} \cup \{a; b; x\} = N(x) \setminus \{a; b\} \cup \{a; b; x\} = N[x];$$

as required. □

4.8.3 Threshold Graphs

Recall from Section 2.3 that threshold graphs are graphs which can be partitioned into a clique and an independent set, where the neighbourhoods of the independent set can be ordered by inclusion. In Gorzny [67], there was an attempt to show that Imbalance has a linear time solution on threshold graphs, which is also unfortunately not correct. We first formally define a threshold partition in the following subsection. Afterwards, in the next subsection, we discuss what went wrong in that proof.

Threshold Partition Definition

We first define a threshold partition before stating some properties of a threshold partition.

Let $(C; I)$ be a split partition where C is a clique and I is an independent set; among all possible choices, choose a split partition that maximizes the cardinality of I. We may assume that every vertex $c \in C$ has a neighbour in I, as otherwise we may take a new split partition $(C'; I')$ where $C' = C \setminus \{c\}$ and $I' = I \cup \{c\}$ for any $c \in C$ that does not have a neighbour in I, which would have a larger independent set. We will partition I into sets $(I_0; I_1; \ldots; I_\ell)$ such that I_0 is the set of isolated vertices, and $N(I_1) \subset N(I_2) \subset \cdots \subset N(I_\ell)$, where ℓ is largest possible. All vertices in I_j have the same neighbours (and therefore the same degree) for $0 \leq j \leq \ell$. The partition on I defines a partition $(C_1; C_2; \ldots; C_\ell)$ of C, where $C_1 = N(I_1)$ and $C_j = N(I_j) \setminus N(I_{j-1})$ for $1 \leq j \leq \ell$. All vertices in C_j have the same

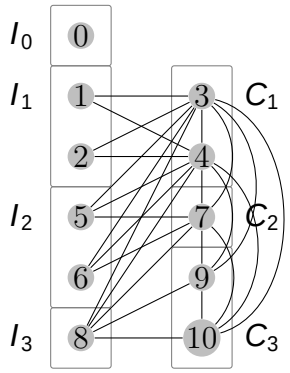


Figure 4.13: A threshold graph G with the levels of a threshold partition indicated.

degree for 1 j `. Vertices of $C_i$ are on the $i$th *level* of the clique, and vertices of $I_i$ are on the $i$th *level* of the independent set. By construction, each set $I_i$ and $C_i$ is non-empty for 1 $i$ `. Figure 4.13 shows an example of a threshold graph with a threshold partition. The following observation is immediate from the definition of a threshold partition.

Observation 4.8.4. *Let G be a threshold graph with a threshold partition $(C; I)$. If $j > i$, then*

1. *$N[C_j]$ ($N[C_i]$ and therefore $jN[C_i]j > jN[C_j]j$; and*
2. *$N(I_j)$ ($N(I_i)$ and therefore $jN(I_i)j > jN(I_j)j$.*

Mahadev and Peled [106] provide a linear time algorithm to determine if a graph is a threshold graph and compute a threshold partition if it is.

What Went Wrong

In Gorzny [67], there was an attempt to show that Imbalance has a linear time solution on threshold graphs.

The claim in that paper is that there is an imbalance-minimal ordering of any threshold graph which does not have any *inverted pairs*, which we now define. We use the following definitions for a connected threshold graph with threshold partition $(C; I) = (C_1 [C_2 [[C`; I_1 [ I_2 [ [ I`)$ and an ordering of the graph . An *inverted-clique-pair* is a pair $(x; y)$ where $y <$ x, $x 2 C_i$, $y 2 C_j$, and $j > i$. An *inverted-independent-pair* is a pair

$(x;y)$ where $y < x$, $x \in I_i$, $y \in I_j$, and $j > i$. An *inverted pair* is an inverted-clique-pair or an inverted-independent-pair.

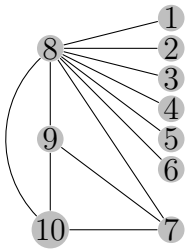
The conjecture is that there is an imbalance-minimal ordering of a threshold graph with no inverted pairs. If there is such an imbalance-minimal ordering for every threshold graph, then this may also be the ordering which minimizes the cutwidth of the graph as well. Cutwidth-minimal orderings of threshold graphs with this property are established by Heggernes et al. [77].

The proof attempts to first show that there an imbalance-minimal ordering of a threshold graph with no inverted-clique-pairs. Then, an imbalance-minimal ordering with no inverted pairs is claimed to be found by simply using an algorithm, Median Placement, of Biedl et al. [8]. Since the threshold partition can be found in linear time, and the Median Placement algorithm runs in linear time, the result would follow. Both steps may be true; however, there is a problem with the first step.

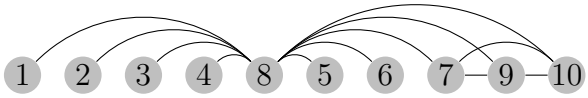
We now describe how the attempted proof attempts fails. First, the proof uses Theorem 4.4.1 to find an imbalance-minimal ordering of the threshold graph G where the independent set of the threshold partition has every vertex perfectly balanced. This allows us to establish that if there is an inverted-clique-pair, then there is one that is "rightmost" in some sense, and moreover, that the neighbourhoods to the right of both vertices in the pair are equal. Then the proof invokes a version of Lemma 4.2.13, which says that the imbalance of at least one of the vertices in the pair would be better if it was swapped past the other, which would also remove the clique pair. However, since the proof does not establish that the two vertices of the inverted pair are consecutive in the ordering, this may increase the imbalance of the vertices in between the pair of vertices.

Helpful Facts and Ideas for a New Proof

It is tempting to try to use the algorithm for Cutwidth on threshold graphs to obtain an imbalance-minimal ordering, given the similarity of the problems. The algorithm for Cutwidth, MinCut (Heggernes et al. [77]), is not described in this thesis but the result of its execution on a graph is shown in Figure 4.14. Figure 4.14 shows that the ordering produced by MinCut does not always meet the requirements of Lemma 3.4.2. However, the algorithm did produce an imbalance-minimal ordering of the graph G. In other small cases, Lemma 3.4.2 can be used on the result of MinCut order to show that the result is in fact imbalance-minimal in addition to being cutwidth-minimal. Therefore, while I conjecture that MinCut does provide imbalance-minimal orderings, it is not proven to do



(a) A threshold graph G.



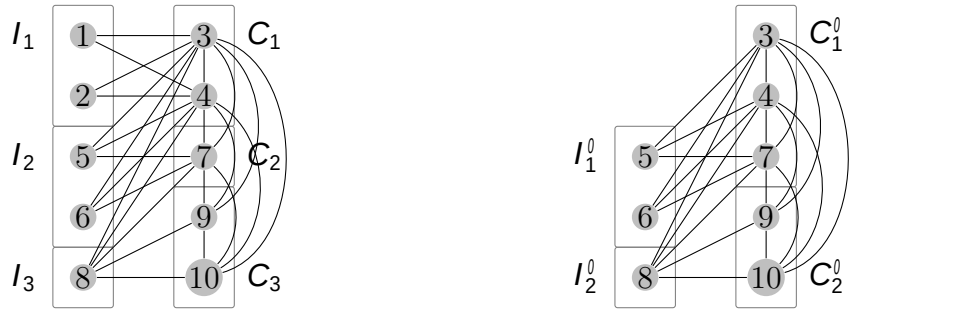
rank (v) 1 1 1 1 1 1 1 1 1 3

(b) An ordering of G generated by the algorithm MinCut for Cutwidth on threshold graphs (Heggernes et al. [77]). Note that $cw(G) = cw(\ ) = 5$ but $im(G) = im(\ ) = 12 > 2cw(G) = 10$. Also observe that there is no single point where the ranks of the vertices in change sign; such an ordering exists $(h1; 2; 3; 4; 8; 7; 9; 10; 5; 6i)$, but has a larger cutwidth.

Figure 4.14: A threshold graph G where $im(G) > 2cw(G)$.

so. Moreover, the graph in Figure 4.14 shows that for a threshold graph G, it is not always the case that $im(G) = 2cw(G)$.

An alternative proof may exploit the recursive linear structure of threshold graphs which is presented by the next lemma. A proof by induction on the number of levels of the threshold graph may be possible. In such a proof, the base case would have one level; it's not possible to have any inverted pairs. For the induction step, one can observe that $I_1 = U(G)$ is also present in $G \setminus I_1$, and could be considered the leftmost set of clique vertices in the threshold partition of $G \setminus I_1$. Then, all that remains is to modify the ordering provided by a suitable induction hypothesis to include the vertices of I_1 so that the imbalance of the ordering is minimum. This has some promise for a couple of reasons. First, Lemma 4.4.3 says that one knows there is always an ordering of G where the vertex $v \in I_1$ which is to be added to the ordering may come first. This may let one fill an ordering with vertices from I_1 in a left-to-right manner. Second, Lemma 4.2.10 gives one a lower bound for the imbalance of the graph for $G \quad v$. However, in our attempt of this proof, the lower bound is not always tight (in particular, when $|U(G)| > 1$), and thus more study is necessary. However, much of the heavy lifting, or the inspiration for it, may already be done.



(a) A threshold graph G. (b) The graph G_2 constructed from G.

Figure 4.15: An example of the recursive structure of threshold graphs.

The following lemma is helpful to understand the recursive structure of threshold graphs.

Lemma 4.8.5. *If G is a threshold graph on $\ell - 2$ levels, then $G' = G \setminus I_1$ is a threshold graph on $\ell - 1$ levels.*

Proof. It suffices to provide a threshold partition for G'. Partition G' into $(C'; I')$ where

- $C' = C_1', C_2', \ldots, C'_{\ell-1}$ and for $2 \le j \le \ell - 1$, $C_j' = C_{j+1}$, while $C_1' = C_1 \cup C_2$.
- $I' = I_0', I_1', I_2', \ldots, I'_{\ell-1}$ and for $2 \le j \le \ell - 1$, $I_j' = I_{j+1}$, while $I_1' = I_2$ and $I_0' = I_0$.

We verify that this is a valid threshold partition:

- For $v \in I_1' = I_2$, $N_G(v) = N_{G'}(v) = C_1 \cup C_2 = C_1'$.
- For $v \in I_j$ for $2 < j \le \ell$, $N_G(v) = N_{G'}(v) = C_j = C'_{j-1}$.
- For $v \in I_0 = I_0'$, $N_G(v) = N_{G'}(v) = \emptyset$.

□

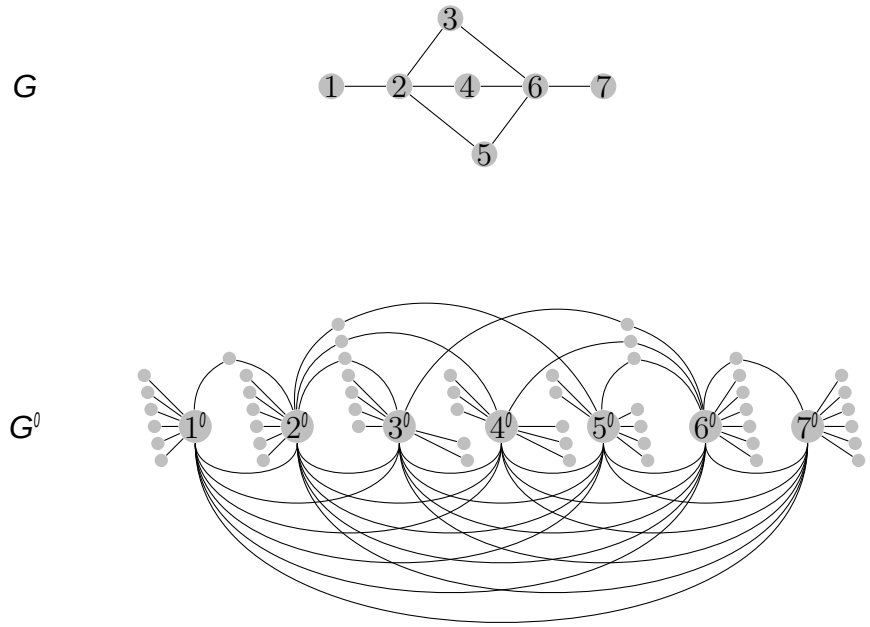


Figure 4.16: A graph illustrating how the NP-complete reduction for Imbalance on split graphs fails. The graph G (top) is transformed into the graph G' (bottom) in the reduction. The set $V(G)$ becomes a clique, and each edge in $E(G)$ is split by a vertex. Lastly, each vertex in $V(G)$ is attached to n 1 pendants intended to "offset" the edges introduced by making $V(G)$ a clique. An imbalance-minimal ordering of either graph is obtained by taking the vertices from left-to-right as drawn in these illustrations.

4.8.4 Split Graphs

Finally, we attempted (Gorzny and Buss [68]) to show that Imbalance is NP-complete for split graphs. Therese Biedl has found a counter-example to the reduction, which is redrawn in Figure 4.16. The set $V(G)$ becomes a clique, and each edge in $E(G)$ is split by a vertex. Lastly, each vertex in $V(G)$ is attached to n 1 pendants intended to "offset" the edges introduced by making $V(G)$ a clique. The reduction incorrectly claims that G has imbalance at most k if and only if $G^{\emptyset}$ has imbalance at most $k+n(n$ 1) where $n = jV(G)j$. The graph $G^{\emptyset}$ on the bottom has imbalance $2 + 7(6) = 2 + n(n$ 1), implying that the graph G has imbalance at most 2. However, the graph G in the figure has imbalance at least 4 since there are four odd degree vertices. The claim that the pendants (only) offset the edges introduced by making $V(G)$ a clique is incorrect.

Chapter 5

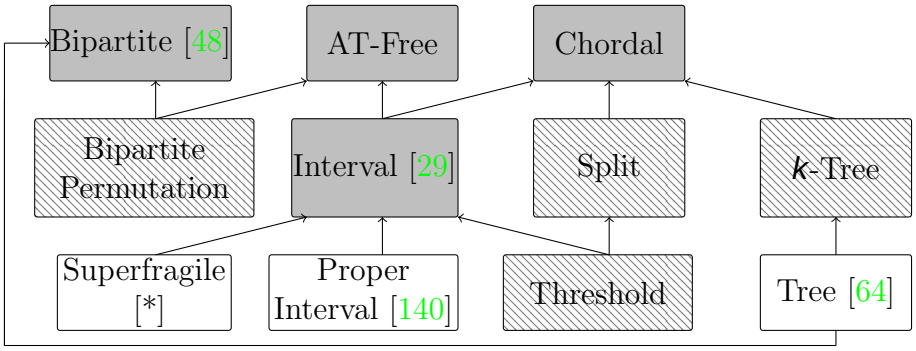
Optimal Linear Arrangement

5.1 Introduction

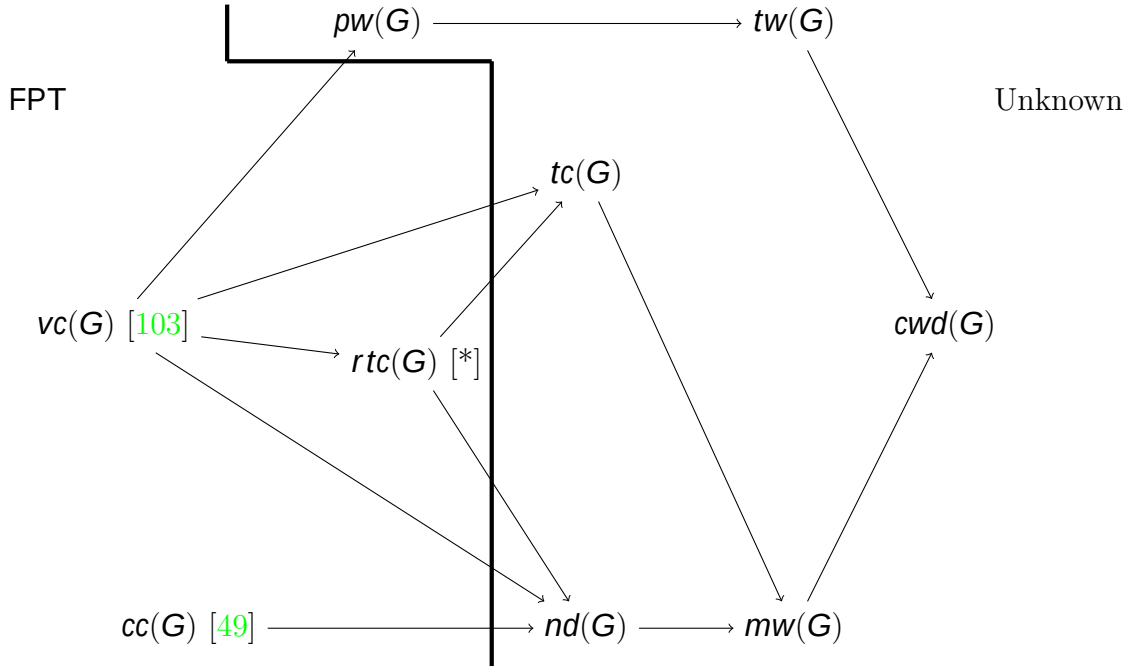
Optimal Linear Arrangement was initially studied due to its various applications and relation to the so-called Bandwidth problem. Optimal Linear Arrangement arises in the context of blackboard wiring, a.k.a. module placement (Adolphson and Hu [1], Hanan and Kurtzberg [73]), data arrangement and ordering (Iordanskii [82]), and error-correcting codes (Harper [74]). Optimal Linear Arrangement is also related to the Bandwidth problem (formally defined in Section 4.6), which asks to minimize the maximum weight of any edge, rather than the sum of all edge weights, in an ordering. Bandwidth is notoriously hard—it is NP-complete even on caterpillars (Monien [112])—but it does have some positive results: Bandwidth is in P for bipartite permutation graphs (Heggernes et al. [76]), chain graphs (Kloks et al. [89]), and interval graphs (Kratsch [91]).

Optimal Linear Arrangement is NP-complete on interval graphs, permutation graphs, cocomparability graphs (Cohen et al. [29]) and bipartite graphs (Even [48]). It is solvable in linear time on proper interval graphs (Yuan and Zhou [140]). It can be solved efficiently on trees (Chung [26], Goldberg and Klipker [64], Shiloach [127]) and small graph classes like hypercubes or grids (see, e.g., Díaz et al. [44]). It can be approximated within $O(\log n)$ on general graphs (Rao and Richa [121]).

Some previous complexity results for Optimal Linear Arrangement are shown in Figure 5.1.



(a) Some known complexity results for Optimal Linear Arrangement on restricted graph classes. The problem is NP-complete for classes with a solid gray background, has unknown complexity for classes with a hatched background, and is in P otherwise. Results for classes marked with [*] are shown in this work.



(b) Some known parameterized complexity results for Optimal Linear Arrangement for common graph parameters. For each parameter to the left of thick line, there is an FPT algorithm for Optimal Linear Arrangement with that parameter. The complexity of Optimal Linear Arrangement is open for those parameters to the right of the thick line. FPT algorithms for parameters marked with [*] are shown in this work.

Figure 5.1: Some known complexity results for Optimal Linear Arrangement.

Summary of Results

In this chapter, we show similarities between Optimal Linear Arrangement and ordering problems: Imbalance and Cutwidth. The previous results for Cutwidth helped establish complexity results for Imbalance, and now we show that those results for Imbalance can be adapted for similar results on Optimal Linear Arrangement. Answering a question of Lilleeng [99], we show that Optimal Linear Arrangement has a polynomial time algorithm on superfragile graphs using a result of Fellows et al. [49]. Fellows et al. [49] observed that there are optimal orderings where equivalence classes of true twins appear together for any graph. We observe that for superfragile graphs there is a solution which is also imbalance-minimal, and therefore can be solved using the results of Chapter 4. We also show that Optimal Linear Arrangement admits an FPT algorithm for graphs with small restricted twin cover number, adapting our approach from Section 3.7 to the integer quadratic program that is known for Optimal Linear Arrangement on graphs with a bounded vertex cover number.

5.2 Preliminaries

The following two results are helpful. The first provides a way to partition a graph's edges to get lower bounds for Optimal Linear Arrangement, while the second is the analogous version of Theorems 3.5.1 and 4.4.2 for Optimal Linear Arrangement.

Lemma 5.2.1 (Corollary 1, Cohen et al. [29]). *Let $G = (V; E)$, $V = V_1 [\quad [V_k$ and $E = E_1 [\quad [E_k$ where $E_1; : : : ; E_k$ are pairwise disjoint. Then $W(G) \quad W(G_1) + : : : + W(G_k)$, where $G_i = (V_i; E_i)$, $1 \quad i \quad k$.*

Theorem 5.2.2 (Lemma 4.2, Fellows et al. [49]). *For any graph G, there exists an optimal linear arrangement (ordering) $\quad$ of $V(G)$ such that each equivalence class of true twins appears consecutively in $\quad$.*

The next lemma connects Imbalance to Optimal Linear Arrangement. It shows that balancing a universal set of vertices within a graph reduces the weights of edges incident with those vertices. Recall that $U(S)$ denotes the set of universal vertices for $S \quad V$, i.e., $U(S) = fv \; j \; v \; 2 \; S$ and $S \quad N[v]g$.

Lemma 5.2.3. *Suppose that $H = (V; E)$ is a graph where $U(H)$ ~~6=~~ ;. Let $G = (V; E^0)$ where $E^0 = E \; n \; E(V \; n \; U(H); V \; n \; U(H))$; that is, G is obtained by deleting all edges from*

H except those with an endpoint in $U(H)$. Note that $U(G) = U(H)$. Finally, suppose that $= hL \quad U(G) \quad Ri$ is an ordering of G for some sets $L; R$ where $jLj > jRj + 1$.

If $^{0} = hL^{0} \quad U(G) \quad R^{0}i$ is an ordering of G for some sets $L^{0}; R^{0}$ where $jL^{0}j = jLj \quad 1$ and $jR^{0}j = jRj + 1$, then $W(\ ) > W(\ ^{0})$.

Proof. Recall that $E(X; Y)$ indicates the edges which have one endpoint in X and the other in Y. The weights of the edges with both endpoints in $U(G)$ do not change as $U(G)$ is a set of true twins, and we may assume the ordering of $U(G)$ is the same in both and ^{0} (Observation 2.1.1). We therefore only consider the weights of edges in $E(U(G); G \quad U(G))$.

Let k be the weights of edges in $E(U(G); U(G))$. Then $W(\ ) \quad k$ represents the weights of the edges in $E(U(G); G \quad U(G))$ in and $W(\ ^{0}) \quad k$ represents the weights of edges in $E(U(G); G \quad U(G))$ in ^{0}.

For convenience, let $r = jRj$, $` = jLj$, $r^{0} = jR^{0}jR$, and $`^{0} = jL^{0}j$. By assumption that $jLj > jRj + 1$, we have that $` > r + 1$, $`^{0} = ` \quad 1$, and $r^{0} = r + 1$.

We can count the weights of the edges in $E(U(G); G \quad U(G))$ in . If we count the weight of the edges by their left endpoints, we have

$$\begin{aligned} W(\ ) \quad k &= \sum_{i=1}^{`} \sum_{j=1}^{jU(G)j} j(` + j) \quad ij + \sum_{i=jU(G)j+`+1}^{r} \sum_{j=1}^{jU(G)j} j(` + j) \quad ij \\ &= \sum_{i=1}^{`} \sum_{j=1}^{jU(G)j} ((` + j) \quad i) + \sum_{i=jU(G)j+`+1}^{r} \sum_{j=1}^{jU(G)j} j(` + j) \quad ij: \end{aligned} \tag{5.1}$$

where we can drop the absolute values in the first double sum since $i \quad `$. The first double sum counts the weights of the edges incident with vertices to the left of $U(G)$ in and $U(G)$, while the second double sum counts the weights of the edges incident with vertices to the right of $U(G)$ in and $U(G)$.

For the edges with an endpoint to the right of $U(G)$ in , we can also count the weights by the right endpoint of each edge, in which case we have

$$\begin{aligned} \sum_{i=jU(G)j+`+1}^{r} \sum_{j=1}^{jU(G)j} j(` + j) \quad ij &= \sum_{i=1}^{`} \sum_{j=1}^{jU(G)j} j(r + j) \quad ij \\ &= \sum_{i=1}^{`} \sum_{j=1}^{jU(G)j} ((r + j) \quad i); \end{aligned} \tag{5.2}$$

where we can again drop the absolute value from the second double sum since $i \quad r$.

Similarly, we can count the weights of the edges in $E(U(G); G \quad U(G))$ in ^{0}. If we count the weight of the edges by their left endpoints, we have

$$\begin{aligned} W(\ ^{0}) \quad k &= \sum_{i=1}^{\grave{}^{0}} \sum_{j=1}^{jU(G)j} j(\grave{}^{0}+j) \quad ij + \sum_{i=jU(G)j+\grave{}^{0}+1}^{r} \sum_{j=1}^{jU(G)j} j(\grave{}^{0}+j) \quad ij \\ &= \sum_{i=1}^{\grave{}^{0}} \sum_{j=1}^{jU(G)j} ((\grave{}^{0}+j) \quad i) + \sum_{i=jU(G)j+\grave{}^{0}+1}^{r} \sum_{j=1}^{jU(G)j} j(\grave{}^{0}+j) \quad ij: \end{aligned} \tag{5.3}$$

where we can drop the absolute values in the first double sum since $i \quad \grave{}^{0}$.

Again, we can also count the weights of edges with an endpoint to the right of $U(G)$ by their right endpoints in ^{0}. In this case, we have

$$\begin{aligned} \sum_{i=jU(G)j+\grave{}^{0}+1}^{r} \sum_{j=1}^{jU(G)j} j(\grave{}^{0}+j) \quad ij &= \sum_{j=1}^{jU(G)j} j(r^{0}+j) \quad ij \\ &= \sum_{j=1}^{jU(G)j} ((r^{0}+j) \quad i); \end{aligned} \tag{5.4}$$

where we can again drop the absolute value from the second double sum since $i \quad r^{0}$.

Putting these equations together, along with some algebra, gives us the following:

$$\begin{aligned} & W(\ ) \quad k \\ &= \sum_{i=1}^{\grave{}} \sum_{j=1}^{jU(G)j} ((\grave{}+j) \quad i) + \sum_{i=jU(G)j+\grave{}+1}^{r} \sum_{j=1}^{jU(G)j} j(L+j) \quad ij \text{ by 5.1} \\ &= \sum_{i=1}^{\grave{}} \sum_{j=1}^{jU(G)j} ((\grave{}+j) \quad i) + \sum_{i=1}^{r} \sum_{j=1}^{jU(G)j} ((r+j) \quad i) \text{ by 5.2} \\ &= \sum_{j=1}^{jU(G)j} ((\grave{}+j) \quad 1) + \sum_{i=2}^{\grave{}} \sum_{j=1}^{jU(G)j} ((\grave{}+j) \quad i) + \sum_{i=1}^{r} \sum_{j=1}^{jU(G)j} ((r+j) \quad i) \\ &> \sum_{j=1}^{jU(G)j} (((r+1)+j) \quad 1) + \sum_{i=2}^{\grave{}} \sum_{j=1}^{jU(G)j} ((\grave{}+j) \quad i) + \sum_{i=1}^{r} \sum_{j=1}^{jU(G)j} ((r+j) \quad i) \end{aligned}$$

$$
\begin{aligned}
&\text{by assumption}\\
&= \sum_{j=1}^{jU(G)j} ((r^{0}+j) \quad 1) + \sum_{i=2}^{\grave{} } \sum_{j=1}^{jU(G)j} ((\grave{}+j) \quad i) + \sum_{i=1}^{r} \sum_{j=1}^{jU(G)j} ((r+j) \quad i)\\
&\text{by assumption}\\
&= \sum_{j=1}^{jU(G)j} ((r^{0}+j) \quad 1) + \sum_{i=1}^{\grave{}\,1} \sum_{j=1}^{jU(G)j} ((\grave{}+j) \quad (i+1)) + \sum_{i=1}^{r} \sum_{j=1}^{jU(G)j} ((r+j) \quad i)\\
&= \sum_{j=1}^{jU(G)j} ((r^{0}+j) \quad 1) + \sum_{i=1}^{\grave{}^{0}} \sum_{j=1}^{jU(G)j} ((\grave{}^{0}+j) \quad i) + \sum_{i=1}^{r} \sum_{j=1}^{jU(G)j} ((r+j) \quad i)\\
&\text{by assumption}\\
&= \sum_{j=1}^{jU(G)j} ((r^{0}+j) \quad 1) + \sum_{i=1}^{\grave{}^{0}} \sum_{j=1}^{jU(G)j} ((\grave{}^{0}+j) \quad i) + \sum_{i=2}^{r\,1} \sum_{j=1}^{jU(G)j} ((r+j) \quad (i \quad 1))\\
&= \sum_{j=1}^{jU(G)j} ((r^{0}+j) \quad 1) + \sum_{i=1}^{\grave{}^{0}} \sum_{j=1}^{jU(G)j} ((\grave{}^{0}+j) \quad i) + \sum_{i=2}^{r^{0}} \sum_{j=1}^{jU(G)j} ((r^{0}+j) \quad i)\\
&\text{by assumption}\\
&= \sum_{i=1}^{\grave{}^{0}} \sum_{j=1}^{jU(G)j} ((\grave{}^{0}+j) \quad i) + \sum_{i=1}^{r^{0}} \sum_{j=1}^{jU(G)j} ((r^{0}+j) \quad i)\\
&= \sum_{i=1}^{\grave{}^{0}} \sum_{j=1}^{jU(G)j} j(\grave{}^{0}+j) \quad ij + \sum_{i=jU(G)j+\grave{}^{0}+1}^{r} \sum_{j=1}^{jU(G)j} j(\grave{}^{0}+j) \quad ij \text{ by 5.4}\\
&= W(\ ^{0}) \quad k \text{ by 5.3,}
\end{aligned}
$$

as required. □

Lemma 5.2.4. *Suppose that $H = (V;E)$ is a graph where $U(H) \neq ;$. Let $G = (V;E^{0})$ where $E^{0} = E \cap E(V \cap U(H);V \cap U(H))$; that is, G is obtained by deleting all edges from H except those with an endpoint in $U(H)$. Note that $U(G) = U(H)$. Finally, suppose that $= hL \quad U(G) \quad Ri$ is an ordering of G for some sets $L;R$ where $jLj < jRj + 1$.*

If $^{0} = hL^{0} \quad U(G) \quad R^{0}i$ is an ordering of G for some sets $L^{0};R^{0}$ where $jL^{0}j = jLj + 1$ and $jR^{0}j = jRj \quad 1$, then $W(\ ) > W(\ ^{0})$.

Proof. The result is immediate from Lemma 5.2.3 applied to ^{R}. □

We also require the following result as well, which is a closed formula for the weight of a complete graph.

Theorem 5.2.5 (Remark 2, Liu and Williams [102])**.** *For a complete graph K_n,*

$$W(K_n) = \binom{n+1}{3} :$$

5.3 Superfragile Graphs

Lilleeng [99] asked if Optimal Linear Arrangement has a polynomial time algorithm for superfragile graphs. We prove the following corollary of Theorem 5.2.2, answering the question positively, using the same approach as in Section 4.5.

Recall that by Observation 2.3.1, if G is a connected superfragile graph which is not a complete graph, then G was constructed by a complete join of U and the graph consisting of the cliques $C_1;:::;C_k$. The $k+1$ equivalence classes of true twins in G are the sets $C_1;:::;C_k$ along with the set U.

Theorem 5.3.1. *If G is a superfragile graph, then $W(G)$ can be computed in $O(n^2)$ time.*

Proof. We may assume $G = (V;E)$ is connected; if not, we can compute the weight of each component and add them together. If G is a complete graph, the result follows by Theorem 5.2.5. If G is not a clique, $G \quad U(G)$ is a disjoint union of cliques $C_1;:::;C_k$ (Observation 2.3.1). Each C_i is an equivalence class of true twins in G. Therefore by Theorem 5.2.2 there is an minimum weight ordering of G where the twins of each C_i appear consecutively. Since $U(G)$ is a also a class of true twins (Observation 2.1.1), $U(G)$ would appear together in one such as well.

By Lemma 5.2.3, the weights of the edges in $E(U(G);V \quad U(G))$ decrease as $U(G)$ becomes more balanced. Therefore, we need to minimize the *imbalance* of $U(G)$ so that the weights of the edges between $U(G)$ and $\bigcup_{i=1}^{k} C_i$ are minimized (edges within each C_i are already minimized). This can be done by solving Imbalance on G using the algorithm of Theorem 4.5.2. Specifically, set $a_i = jC_ij$ for $1 \quad i \quad k$. An $O(N \quad A)$-time algorithm is known for this problem (see, e.g., Mertens [109]), where N is the size of the list and A bounds each element in the list. Taking $N = A = n$ means we can solve this instance in $O(n^2)$ time. Given a solution A, we can create an imbalance-minimal ordering as

follows. Starting with an empty ordering, for each $a_i \in A$, append all the vertices of C_i in any order to the existing ordering. Then, append all the vertices of $U(G)$ in any order. Finally, for each $a_i \in A$, append all the vertices of C_i consecutively. Take the resulting order to be : each equivalence class of true twins is consecutive, and the imbalance of $U(G)$ is minimized. The result is also an optimal linear arrangement for G. □

5.4 Restricted Twin Cover Number Parameterization

In this section, we show that there is an FPT algorithm for Optimal Linear Arrangement when the parameter is the restricted twin cover number. We use the FPT algorithm of Lokshtanov [103] for graphs of bounded vertex cover as inspiration.

We formulate Optimal Linear Arrangement as an instance of the following problem.

Problem 5.4.1 (Integer Quadratic Programming). *Let matrices $Q \in \mathbb{Z}^{n \times n}$, $A \in \mathbb{Z}^{m \times n}$, and $b \in \mathbb{Z}^{m \times 1}$ be given. Find a vector $x \in \mathbb{Z}^{n \times 1}$ that minimizes the objective function $x^T Q x$ and satisfies the m inequalities, that is, $Ax \leq b$.*

An instance of Integer Quadratic Programming, called an *integer quadratic program* (IQP), may be infeasible, unbounded, or not unbounded. An infeasible instance is one where no vector x satisfies $Ax \leq b$, whereas an unbounded instance is one where for every integer y, there is some vector x such that $x^T Q x \leq y$. A not unbounded instance is one which is feasible (there is a vector x that satisfies $Ax \leq b$) and there is a vector x such that $x^T Q x$ is in fact minimum over all feasible vectors.

Lokshtanov [103] showed the following result, establishing that Integer Quadratic Programming has an FPT algorithm when the parameter is the number of variables and the largest absolute value of the entries in Q and A.

Theorem 5.4.2 (Theorem 1, Lokshtanov [103]). *There exists an algorithm that given an instance of Integer Quadratic Programming, runs in time $f(n,)L^{O(1)}$, and outputs a vector $x \in \mathbb{Z}^n$. Here, L is the total number of bits required to encode the input IQP, n is the number of variables in the IQP, and is the largest absolute value of the entries in Q and A. If the input IQP has a feasible solution then x is feasible, and if the input IQP is not unbounded, then x is an optimal solution.*

We modify the following integer quadratic program (IQP) formulation for Optimal Linear Arrangement on graphs of bounded vertex cover number by Lokshtanov [103].

The notation is updated to match the notation used in this work, where possible. Recall that $I_S = \{v \in I : N(v) = S\}$ for a set S. As in the case of Cutwidth, the input to Optimal Linear Arrangement parameterized by the vertex cover number of the graph will be a graph G and a vertex cover C of G such that $|C| \leq k$. If C is not provided but $vc(G) \leq k$, then C can be found in time $O(2^k n)$ (see, e.g., Theorem 3.2.1, Downey and Fellows [46]).

IQP Formulation 5.4.3 (Optimal Linear Arrangement, Lokshtanov [103])**.**

We assume that a vertex cover C of G of size at most k is given as input. The remaining set of vertices $I = V(G) \setminus C$ forms an independent set. Furthermore, I can be partitioned into at most 2^k sets I_S based on each subset $S \subseteq C$.

Let $C = \{c_1, c_2, \ldots, c_k\}$. By trying all $k!$ permutations of C we may assume that the optimal solution σ satisfies $\sigma(c_i) < \sigma(c_{i+1})$ for every $1 \leq i \leq k-1$. For every i between 1 and $k-1$, we define the location L_i of σ to be the set of vertices appearing between c_i and c_{i+1} according to σ. Location L_0 is the set of all vertices appearing before c_1 and location L_k is the set of vertices appearing after c_k. For every location L_i and neighbourhood $S \subseteq C$ of vertices we denote by I_S^i the set $L_i \cap I_S$ of vertices of neighbourhood S appearing in location i.

We say that an ordering σ is *homogeneous* if, for every location L_i and every neighbourhood $S \subseteq C$ the vertices of I_S^i appear consecutively in σ.

Fellows et al. [49] showed that there exists an optimal solution which is homogeneous, and where in every location, the vertices are ordered from left to right in non-decreasing order by their rank. We will call such an ordering solution *super-homogeneous*.

Notice that a super-homogeneous linear arrangement σ is completely defined (up to swapping positions of vertices with the same neighbourhood) by specifying for each i and each neighbourhood $S \subseteq C$ the size $|I_S^i|$. For each location i and each neighbourhood S we introduce a variable $x_S^i \in \mathbb{Z}$ representing $|I_S^i|$. Clearly the variables x_S^i need to satisfy

$$\forall i \leq k; \forall S \subseteq C \quad x_S^i \geq 0 \tag{5.5}$$

and

$$\forall S \subseteq C \quad \sum_{i=0}^{k} x_S^i = |I_S|; \tag{5.6}$$

On the other hand, every assignment to the variables satisfying these (linear) constraints corresponds to a super-homogeneous linear arrangement of G with $|I^i_S| = x^i_S$ for every neighbourhood S and location i.

We now analyze the cost of as a function of the variables. The goal is to show that $W(\ )$ is a quadratic function of the variables with coefficients that are bounded from above by a function of k. The coefficients of this quadratic function are not integral, but *half-integral*, namely integer multiples of 1=2. For the analysis below is helpful to re-write $W(\ )$. For a fixed ordering of the vertices we say that and edge uv *ies over* the vertex w if

$$\min(\ (u);\ \ (v)) < \ (w) < \max(\ (u);\ \ (v)):$$

We define the "fly over" relation for edges and vertices, i.e., uv w means that uv flies over w. Since an edge with $(u) < (v)$ flies over the (v) (u) 1 vertices appearing between (u) and (v) it follows that

$$W(\ ) = |E(G)| + \sum_{uv \in E(G)} \sum_{\substack{w \in V(G) \\ uv \ \ w}} 1:$$

we partition the set of edges of G into several subsets as follows. The set E_C is the set of all edges with both endpoints in C. For every location i with $i \in \{0;\dots;k\}$, every $j \in \{1;\dots;k\}$ and every $S \subseteq C$ we denote by $E^S_{i;j}$ the set of edges whose one endpoint is in I^i_S and the other is c_j. Notice that $|E^S_{i;j}|$ is either equal to x^i_S or to 0 depending on whether vertices of neighbourhood S are adjacent to c_j or not. We have that

$$W(\ ) = |E(G)| + \sum_{c_ic_j \in E(G)} \sum_{\substack{w \in V(G) \\ c_ic_j \ \ w}} 1 + \sum_{i;j;S} \sum_{uc_j \in E^S_{i;j}} \sum_{\substack{w \in V(G) \\ uc_j \ \ w}} 1: \tag{5.7}$$

Further, for each edge $c_ic_j \in E_C$ (with $i < j$) we have that

$$\sum_{\substack{w \in V(G) \\ c_ic_j \ \ w}} 1 = j \quad i \quad 1 + \sum_{p=i}^{j-1} \sum_{S \ \ C} x^p_S: \tag{5.8}$$

In other words, the first double sum of Equation 5.7 is a linear function of the variables. Since $|E_C| \quad \binom{k}{2}$, the coefficients of this linear function are integers upper bounded by $\binom{k}{2}$.

We now turn to analyzing the second part of Equation 5.7. We split the triple sum in three parts as follows

$$\sum_{i,j,S} \sum_{uc_j \in E^S_{i,j}} \sum_{w \in V(G)} \frac{1}{uc_j \quad w} = \sum_{i,j,S} \sum_{uc_j \in E^S_{i,j}} \sum_{w \in C} \frac{1}{uc_j \quad w} + \sum_{i,j,S} \sum_{uc_j \in E^S_{i,j}} \sum_{w \in I^i_S} \frac{1}{uc_j \quad w} + \sum_{i,j,S} \sum_{uc_j \in E^S_{i,j}} \sum_{w \in I^i_S} \frac{1}{uc_j \quad w} : \tag{5.9}$$

For any fixed $i, j,$ and S, and any edge $uc_j \in E^S_{i,j}$ the number of vertices $w \in C$ such that $uc_j \quad w$ depends solely on i and j. It follows that

$$\sum_{uc_j \in E^S_{i,j}} \sum_{w \in C} \frac{1}{uc_j \quad w} = f(i,j) \quad x^i_S \tag{5.10}$$

for some function f, which is upper bounded by k (since $|C| = k$).

Consider a pair of vertices u, w in I^i_S and a vertex $c_j \in C$ such that vertices with the same neighbourhood as u and w are adjacent to c_j. Either the edge uc_j flies over w or the edge wc_j flies over u, but both of these events never happen simultaneously. Therefore,

$$\sum_{uc_j \in E^S_{i,j}} \sum_{w \in I^i_S} \frac{1}{uc_j \quad w} = \frac{x^i_S}{2} = \frac{(x^i_S)^2}{2} \quad \frac{x^i_S}{2}. \tag{5.11}$$

In other words, this sum is a quadratic function of the variables with coefficients $1/2$ and $-1/2$. Further, if vertices in I_S are not adjacent to c_j this sum is 0.

For the last double sum in Equation 5.9 consider an edge $uc_j \in E^S_{i,j}$ and vertex $v \in I^{i'}_{S'}$ such that $S' \neq S$ or $i' \neq i$. If uc_j flies over v then all the edges in $E^S_{i,j}$ fly over all the vertices in $I^{i'}_{S'}$. Let $g(i, j, S, i', S')$ be a function that returns 1 if vertices in I_S are adjacent to c_j and all the edges in $E^S_{i,j}$ fly over all the vertices in $I^{i'}_{S'}$. Otherwise, $g(i, j, S, i', S')$ returns 0. It follows that

$$\sum_{uc_j \in E^S_{i,j}} \sum_{w \in I^i_S} \frac{1}{uc_j \quad w} = x^i_S \sum_{(i',S') \neq (i,S)} g(i, j, S, i', S') x^{i'}_{S'}. \tag{5.12}$$

In other words, this sum is a quadratic function of the variables with 0 and 1 as coefficients.

The outer sum of Equation 5.9 goes over all 2^k choices for S, $k+1$ choices for i, and k choices for j. Since the sum of quadratic function is a quadratic function, this concludes the analysis and proves the following lemma.

Lemma 5.4.4 (Lemma 10, Lokshtanov [103]). *$W(\ )$ is a quadratic function of the variables $\{x_S^i\}$ with half-integral coefficients between $-2^k k^2$ and $2^k k^2$. Furthermore, there is a polynomial time algorithm that given G computes the coefficients.*

For each permutation $c_1, \ldots, c_k$ of C we can make an integer quadratic program for finding the best super-homogeneous solution to Optimal Linear Arrangement which places the vertices of C in the order $c_1, \ldots, c_k$ from left to right. The quadratic program has variable set $\{x_S^i\}$ and constraints as in Equations 5.5 and 5.6. The objective function is the one given by Lemma 5.4.4 but with every coefficient multiplied by 2. This does not change the set of optimal solutions and makes all the coefficients integral. This quadratic program has at most $2^k(k+1)$ variables, $2^k(k+2)$ constraints, and all coefficients are between $-2^{k+1}k^2$ and $2^{k+1}k^2$. Furthermore, since the domain of all variables is bounded the IQP is bounded as well. Thus, we can apply Theorem 5.4.2 to solve each IQP in time $f(k)\ n$.

This completes the description of the IQP for Optimal Linear Arrangement adapted from Lokshtanov [103]. The remainder of this section describes the modifications needed to adapt the IQP to the restricted twin cover number for the main result of this section.

Theorem 5.4.5. Optimal Linear Arrangement *is fixed-parameter tractable when the parameter is the restricted twin cover number of the graph.*

Proof. Let G be a graph with $rtc(G) = |T| + q = k$ for some $q, k \geq 0$ and twin cover T. We can compute T in time $f(k)\ n^{O(1)}$ by Theorem 2.5.19. There are q non-trivial components of $G - T$. We modify the approach of IQP 5.4.3 as follows.

First, instead of trying every permutation of a vertex cover, we try every permutation of the twin cover T; there are $|T|!$ such permutations. Let $Q_1, \ldots, Q_q$ be the non-trivial components of $G - T$, and let $Q = \bigcup_{i=1}^{q} Q_i$. By Lemma 2.5.2, each non-trivial component of $G - C$ is a set of true twins; by Theorem 5.2.2 there is an optimal linear arrangement of G such that each of these sets is consecutive.

Let $G' = (V', E')$ where $V' = V$ and $E' = E(G) \setminus (\bigcup_{i=1}^{q} (Q_i \times (V - Q_i)))$. That is, G' is the graph obtained by deleting the edges of the non-trivial components Q_i of $G - T$. If T is a

twin cover of G, then by Lemma 2.5.1, T is a vertex cover in $G^{\emptyset}$, and therefore also a twin cover.

For each permutation $t_1; : : : ; t_k$ of T, we create $(jTj+1)^q$ instances of an integer quadratic program based on the one described above. Each instance of the program will place the q non-trivial components of $G \quad T$ into locations (i.e., the set of locations L_i for $0 \quad i \quad k+1$) of the ordering imposed on $t_1; : : : ; t_k$, the vertices of T.

In particular, there are $(jTj + 1)^q$ ways to put q sets Q_i into $jTj + 1$ locations L_i $(0 \quad i \quad k)$. For each combination, we let $B(S; i)$ be the union of those components Q_j with $N_G(Q_j)$ placed into bin i. Therefore, we require that each x_S^i is at least the sum of some values $jB(S; i)j$.

Thus, we will create $(jTj!) \ (jTj+1)^q \quad (rtc(G)!) \ (rtc(G)+1)^{rtc(G)}$ instances of an IQP referring to $G^{\emptyset}$ (not G).

As in the IQP used by Lokshtanov, the variable x_S^i represents the number of variables of $G \quad (T \ [\ Q)$ with neighbourhood $S \quad T$. Let Q_S be the union of the non-trivial components of $G \quad T$ that have the neighbourhood $S \quad T$, i.e.,

$$Q_S = \bigcup_{\substack{1 \ j \ q \\ N_G(Q_j)=S}} Q_j:$$

Instead of Equation 5.6 from the Lokshtanov program, we modify it as follows:

$$8S \quad C \quad \sum_{i=0}^{k} x_S^i = jI_Sj + jQ_Sj: \tag{5.13}$$

To each instance of the program, we add the following constraint (instead of Equation 5.5; note that if $B(S; i) = ;$, then the constraint for that pair $(S; i)$ is the same):

$$8i \quad k; 8S \quad C \quad x_S^i \quad jB(S; i)j: \tag{5.14}$$

Finally, we update the objective function. Starting with the objective function of the IQP for graphs with bounded vertex cover number, we add a constant term h which is equal to $\sum_{i=1}^{q} \binom{jQ_ij+1}{3}$, which accounts for the weights of edges between vertices in each component Q_i by Theorem 5.2.5. These edges are not included in $G^{\emptyset}$, so we are not double counting them by adding this term. The term h can be computed prior to creating any instances of the IQP and passed to an instance as a constant each time (that is, we only need to compute each binomial coefficient once).

The reason we update the objective function and introduce the term h is because the original IQP formulation finds a super-homogeneous optimal linear arrangement. In such an ordering, the value h necessarily contributes to the total weight of the ordering. On the other hand, ignoring those edges only, the vertices within a non-trivial component are indistinguishable from a vertex in a trivial component with the same neighbourhood in T. Therefore, when we have x^i_S vertices of this kind within a location L_i, we can partition this set – which is consecutive by definition of super-homogeneous – into the non-trivial components placed into this location in an arbitrary order, followed by the vertices of the non-trivial components. For example, if Q_j and Q_{j+1} are the only non-trivial components of $G \quad T$ for some j placed into location L_i for some i, necessarily $x^i_S \quad jQ_jj + jQ_{j+1}j$. Therefore, within location L_i, we can take the first jQ_jj vertices of x^i_S to be those of Q_j, the next $jQ_{j+1}j$ vertices to be those of Q_{j+1}, and any remaining vertices with the neighbourhood S to be trivial components of $G \quad T$. All the edges from Q_j to T and Q_{j+1} to T will be accounted for, while the $\binom{jQ_jj+1}{3} + \binom{jQ_{j+1}j+1}{3}$ edges within Q_j and Q_{j+1} are accounted for by the h term in the objective function.

Therefore, we have the same number of constraints as in the IQP for graphs with bounded vertex cover number, and no additional variables: each instance of the IQP receives each $jB(S;i)j$ value and $jI_Sj + jQ_Sj$ value as constants. Moreover, the coefficients are not increased in either Q or A (only the right side of the inequalities changes, which is the b vector) and the IQP is still bounded as all variables are still bounded. Taking the ordering with the smallest weight will allow us to compute $W(G)$, since by Theorem 5.2.2 there is an optimal layout where all classes of true twins appear consecutively. □

Part II

Type II Problems

Chapter 6

End-Vertex Problem

6.1 Introduction

The well-known search algorithms presented in Section 2.6 have been applied to a wide range of problems. For example, BFS can be used to check if a graph is bipartite and DFS can be used to find cut vertices of a graph (see, e.g., Bondy and Murty [14]). LBFS is used for recognizing graph classes, computing graph parameters, and detecting certain graph structures (a survey is provided in Corneil et al. [30]). Other searches, like Lexicographic Depth-First Search (LDFS), have also been shown to be helpful for other problems. For example, LDFS is used for computing maximal cardinality matchings on so-called cocomparability graphs (Mertzios et al. [110]).

One of the first applications of end-vertices are *perfect elimination orderings*. A perfect elimination ordering (PEO) for a graph G is an ordering such that for all $v \in V(G)$, $N[v] \setminus v$ induces a clique in G. It is known that a graph is chordal if and only if it has a PEO (Fulkerson and Gross [55]), and that PEOs can be generated efficiently using LBFS. This is because if the input graph for LBFS is chordal then the last visited vertex of an LBFS is simplicial. This resulted in an efficient algorithm for recognizing chordal graphs (see also Rose, Tarjan, and Lueker [124]). However, a perfect elimination ordering can also be found by Maximal Cardinality Search (MCS) (Tarjan and Yannakakis [131, 132]). Understanding the end-vertices generated by these algorithms enabled simple inductive proofs of correctness for such recognition algorithms.

These results have stimulated research on last visited vertices of various search algorithms on many classes of graphs (Berry et al. [6], Berry and Bordat [7], Corneil et al. [33],

Brandstädt et al. [16], Gorzny [66], Gorzny and Huang [69], Jamison and Olariu [83], Zou et al. [141]). However, end-vertices are still not fully understood. For example, not all simplicial vertices of a chordal graph are LBFS end-vertices. Understanding end-vertices may enable these algorithms to be applied to even more problems.

Naturally, subsequent research studied the complexity of determining if a vertex is a possible end-vertex of a search. Recall that this is the S-End-Vertex: given a graph $G = (V; E)$, a vertex $t \ 2 \ V$, and a search algorithm S, determine if there exists an ordering $: V \ ! \ f1; : : : ; ng$ (where $n = jV j$) generated by S of V such that $(t) = n$.

The end-vertices of BFS, DFS, LBFS, LDFS, MNS, and MCS have all been studied on various class graphs (Beisegel et al. [5], Cao et al. [21], Charbit et al. [23], Corneil et al. [33], Gorzny and Huang [69], Zou et al. [141]). Table 6.1 shows the known complexity for S-End-Vertex.

S-End-Vertex is NP-complete in general for each of the aforementioned search algorithms (Beisegel et al. [5], Cao et al. [21], Charbit et al. [23], Corneil et al. [33], Zou et al. [141]). In fact, the problem is NP-complete for *weakly chordal* graphs: graphs without an induced cycle of length at least five.

On restricted graph classes, characterizations of end-vertices enable tractable algorithms for S-End-Vertex. Recall that a vertex v is *admissible* if there are no vertices unrelated to it, that is, for every pair of vertices $u; w$, every $u; w$ path contains a vertex of $N[v]$. Corneil, Olariu and Stewart [36] proved that the end-vertices of LBFS of an interval graph are precisely the simplicial and admissible vertices after establishing that all LBFS end-vertices of AT-free graphs are admissible. The characterization of end-vertices of LBFS of interval graphs led to a linear time solution to the LBFS-End-Vertex for interval graphs. LBFS end-vertices of split graphs, (House, Hole, Domino)-free graphs and distance-hereditary graphs also have nice properties (Charbit et al. [23], Brandstädt et al. [16], Jamison and Olariu [83]). End-vertex characterizations on split graphs have since been found for BFS, LDFS, MNS, and MCS (Beisegel et al. [5], Charbit et al. [23]). On general chordal graphs, end-vertices are understood for LDFS, MNS, and MCS (Beisegel et al. [5], Cao et al. [21]). There is also some preliminary work on determining if there are classes of graphs where several orderings are valid executions of multiple searches simultaneously (Krnc and Pivač [93]).

Recently, Gorzny and Huang [69] proved[1] that LBFS-End-Vertex is NP-complete for bigraphs (see also Kratsch et al. [92]). For the subclass of bipartite permutation graphs, Gorzny and Huang [69] obtained a characterization of end-vertices of LBFS using the

[1] This result is from Jan Gorzny's master's thesis [66], which is why it does not appear in this work.

Class	BFS	DFS	LBFS	LDFS	MNS	MCS
All Graphs	NPC	NPC	NPC	NPC	NPC [5]	NPC [5]
Weakly Chordal	NPC [23]	NPC	NPC [33, 21]	NPC [23]	NPC [5]	NPC [21]
Chordal	?	NPC	?	L [21]	L [5]	P [21]
Interval	L [21]	L [5]	L [33]	L	L [5]	P
Proper Interval	L	L [5]	L [33]	L [5]	L [5]	L [5]
Split	P [23]	NPC [23]	L [5]	L [5]	L [5]	L [5]
Bipartite	NPC [23]	NPC [66]	NPC [69]	NPC [141]	?	NPC [141]
Bipartite Permutation	P [*]	L [*]	P [69] *!* L [*]	?	P [*]	?

Table 6.1: Known complexity results for S-End-Vertex. An entry "L" indicates that there is a linear time solution, an entry "P" indicates that there is a polynomial time solution, and an entry "NPC" indicates that the problem is NP-complete. An arrow *!* indicates that a result has been improved. Results marked with [*] are shown in this work.

notion of eccentricity, which appears in the next section. However, there are vertices which are BFS end-vertices but not LBFS end-vertices (this will be illustrated in Section 6.3), so that result alone does not paint a complete picture of end-vertices for both of these layer searches.

Summary of Results

In this work, we show that both BFS-End-Vertex and MNS-End-Vertex have polynomial time solutions and that DFS-End-Vertex has a linear time solution on bipartite permutation graphs. We also provide a non-trivial linear-time algorithm to determine if a vertex of a bipartite permutation graph is an LBFS end-vertex, which improves the result of Gorzny and Huang [69]. Since bipartite permutation graphs are weakly chordal (recall that they contain no induced cycle of length greater than four), these results demonstrate that there are additional subclasses of weakly chordal graphs for which these problems are tractable.

Our end-vertex results often exploit the same linear structure of the graph. Many results stem from finding a good vertex from which to start a breadth-first search of the graph. We observe that there is a highly structured setup of layered vertices based on the distance from the start vertex along with a good understanding of the components if the closed neighbourhood of the first vertex is removed. In particular, there can only be two "large" components of such a graph, and those can be considered to contain the ends of the linear structure, i.e., admissible vertices. This approach applies even in cases when the search is not related to breadth-first search, like our result for maximal neighbourhood search.

6.2 Preliminaries

We start with some definitions that will be used in this chapter.

> Recall that an $(x;y)$-path P *misses* (or *avoids*) z if $V(P) \setminus N[z] = ;$, that is, P contains neither z nor a neighbour of z; otherwise the vertex z is said to *intercept* (or *hit*) the path P.
>
> A *dominating path* in G is a path P such that no vertex in G is missed by P. A pair of vertices $(x;y)$ is said to be a *dominating pair* if every $(x;y)$-path is a dominating path in G. When $x;y$ are both diametrical and dominating, they are called a *diametrical dominating pair*. These concepts are illustrated in Figure 6.1.

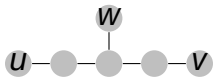


Figure 6.1: An illustration of dominating vertices of a graph. The vertices u and v are a dominating pair of G since every vertex in G intercepts every $(u;v)$-path. Since $d_G(u;v) = \text{diam}(G)$, the pair $(u;v)$ is a diametrical dominating pair. The vertices w and v are *not* a dominating pair of G since u does not intercept the only $(w;v)$-path of G.

Several propositions regarding end-vertices and the structure of bipartite permutation graphs are necessary to simplify many of the proofs in this work. Recall that the diameter of a graph, denoted $\text{diam}(G)$, is the largest distance between any two vertices in the graph and that the eccentricity of a vertex, denoted $\text{ecc}(v)$, is the largest distance between v and any other vertex in the graph.

Proposition 6.2.1 (Combination of Theorem 1, Corneil et al. [32] and Theorem 4.2, Corneil et al. [35])**.** *If v is the end-vertex of an LBFS of an AT-free graph G, then v is admissible and $\text{ecc}(v) \quad \text{diam}(G) \quad 1$.*

Theorem 6.2.2 (Theorem 4.2, Corneil et al. [35])**.** *Let G be a connected AT-free graph and v be an admissible vertex in G. Suppose that there is an LBFS ordering which begins at v and ends at w. Then $v;w$ are a dominating pair in G. Moreover, if $\text{ecc}(v) = \text{diam}(G)$, then $v;w$ are a diametrical dominating pair.*

Fix an arbitrary vertex z and recall that $L_i(z)$ denotes the ith layer of a graph with respect to z and is equal to all vertices at distance i from z. Let $\grave{}$ be a natural number. Note that when G is bipartite, $L_{\grave{}}(z)$ is an independent set for each $\grave{}$. To see this, suppose that this was not the case and $(u;v) \in E(G)$ for $u;v \in L_i(z)$ for some $i > 0$ and $u;v;z \in V(G)$. Then a shortest $(v;z)$-path and a shortest $(u;z)$-path along with the edge $(u;v)$ would form an odd walk, contradicting the fact that the graph is bipartite. We restate this fact so that we can reference it in proofs.

Observation 6.2.3. *If G is a bipartite graph, for any vertex $v \in V(G)$ and $0 \quad i \quad \text{ecc}(v)$, $L_{\grave{}}(v)$ is an independent set.*

Fix an arbitrary vertex z. We shall use $N^z_{\grave{}}(a)$ to denote the set of all neighbours of a in $L_{\grave{}}(z)$, that is, $N^z_{\grave{}}(a) = N(a) \setminus L_{\grave{}}(z)$. It is clear that if $a \in L_{\grave{}+1}(z)$ then $N^z_{\grave{}}(a) \neq \emptyset$.

Lemma 6.2.4 (Lemma 3.3, Gorzny and Huang [69]). *Let G be a bipartite permutation graph and z be a vertex of G. Suppose that C is a connected component of $G \setminus N[z]$ and that $a, b \in L_\ell(z)$ are two vertices in C. Then*

1. *$N^z_{\ell-1}(a) \subseteq N^z_{\ell-1}(b)$ or $N^z_{\ell-1}(a) \supseteq N^z_{\ell-1}(b)$;*
2. *$N^z_{\ell+1}(a) \subseteq N^z_{\ell+1}(b)$ or $N^z_{\ell+1}(a) \supseteq N^z_{\ell+1}(b)$;*
3. *if $N^z_{\ell-1}(a) \subsetneq N^z_{\ell-1}(b)$ then $N^z_{\ell+1}(a) \subseteq N^z_{\ell+1}(b)$;*
4. *if $N^z_{\ell+1}(a) \subsetneq N^z_{\ell+1}(b)$ then $N^z_{\ell-1}(a) \supseteq N^z_{\ell-1}(b)$.*

Suppose that C is a component of $G \setminus N[z]$ and $a, b \in N(z)$. It follows from statement 2 of Lemma 6.2.4 that either $N(a) \setminus C \subseteq N(b) \setminus C$ or $N(a) \setminus C \supseteq N(b) \setminus C$. The next lemma formalizes this idea.

Lemma 6.2.5. *Let C be a component of $G \setminus N[z]$ for some vertex z and bipartite permutation graph G. If $i > 1$, then there is a vertex $c_i \in C \setminus L_i(z)$ such that, if $(L_{i+1}(z) \setminus C) \neq \emptyset$, then $N^z_{i+1}(c_i) = (L_{i+1}(z) \setminus C)$.*

Proof. By Lemma 6.2.4, $N^z_{i+1}(a)$ and $N^z_{i+1}(b)$ are comparable for any $a, b \in L_i(w) \setminus C$ for any component C of $G \setminus N[z]$. Thus we can order the vertices of $L_1(z) \setminus C$ by inclusion based on their neighbours in $L_{i+1}(z) \setminus C$. Therefore there is a vertex which is adjacent to all of $L_{i+1}(z) \setminus C$. □

In particular, if $c \in N(z)$ is a vertex adjacent to the maximum number of vertices in C, then for any $u \in L_\ell(z) \setminus C$ with $\ell \geq 2$, $d(c, u) \geq \ell - 1$. To see this, note that we can construct a path with the desired length by taking the vertices in each layer $L_i(z)$ that have the largest neighbourhood on the next layer $L_{i+1}(z)$ in the component (such vertices always exist by statement 2 of Lemma 6.2.4, and are necessarily adjacent to the entire next layer in the component), until one is adjacent to u.

Recall that a component is trivial if it contains a single vertex and that a non-trivial component is a component which contains an edge (since such a component must have at least two vertices and be connected). Fix an arbitrary vertex z. We call a component C a *deep* component of $G \setminus N[z]$ if it contains an eccentric vertex of z. Note that a deep component of $G \setminus N[z]$ exists if and only if $\text{ecc}(z) \geq 2$. Deep components are particularly relevant for layer search algorithms, while non-trivial components are more useful for non-layer search algorithms. The different kinds of components are illustrated in Figure 6.2.

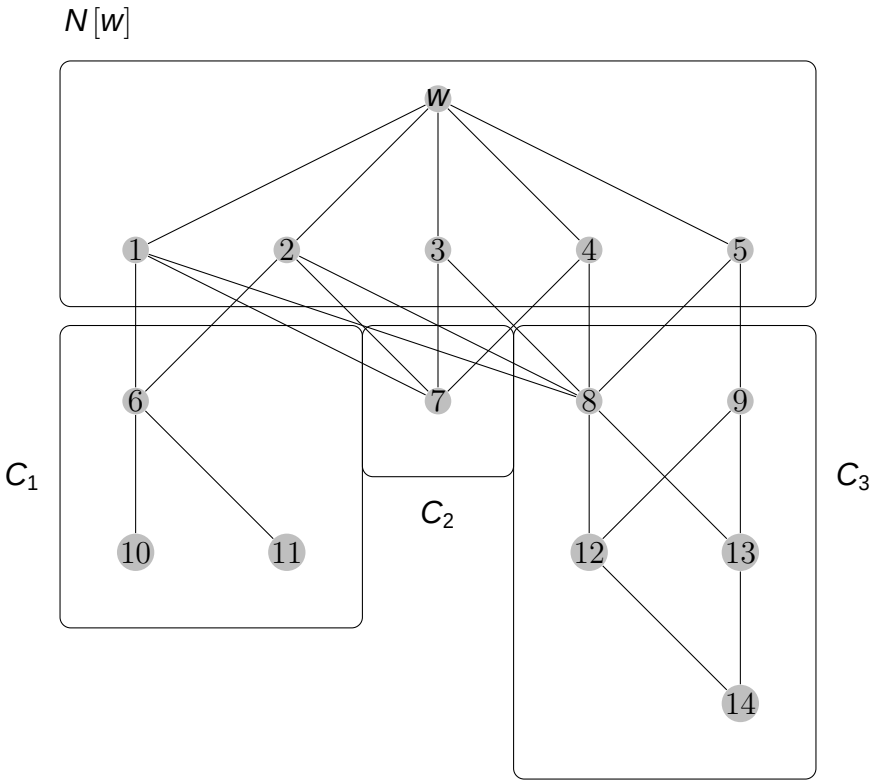


Figure 6.2: An illustration of the types of components considered in this work. The components of $G \quad N[w]$ are $C_1 = f6; 10; 11g$, $C_2 = f7g$ and $C_3 = f8; 9; 12; 13; 14g$. The components C_1 and C_3 are non-trivial components of $G \quad N[w]$, while the component C_2 is trivial. Since $d(14; w) = \text{ecc}(w) = 4$, C_3 is also a deep component.

Lemma 6.2.6 (Lemma 3.4, Gorzny and Huang [69])**.** *Let G be a connected bipartite permutation graph and z be a vertex of G. If $\text{ecc}(z) \quad 3$, then $G \quad N[z]$ has at most two deep components.*

In fact, we can say the following stronger statement.

Lemma 6.2.7. *Let $G = (v; E)$ be a connected bipartite permutation graph and z be a vertex of G. If $\text{ecc}(z) \quad 3$, then $G \quad N[z]$ has at most two non-trivial components.*

Proof. Suppose to the contrary that $G \quad N[z]$ has at least three non-trivial components for a bipartite permutation graph G.

Choose three non-trivial components C_1, C_2, and C_3, of $G \setminus N[z]$. Since each of these has an edge but $L_2(z)$ is an independent set by Observation 6.2.3, there is a vertex $c_i \in (C_i \setminus L_3(z))$ for $1 \leq i \leq 3$.

However, any shortest (c_i, c_j)-path for $i \neq j$, $i, j \in \{1, 2, 3\}$ does not intersect $N_G[c_k]$ for $k \in \{1, 2, 3\} \setminus \{i, j\}$. To see this, consider a shortest (c_i, c_j)-path P which uses as few vertices of C_k as possible. As $c_i \in C_i$, c_i is not adjacent to any vertex of C_k. Similarly, if $c_i' = P \setminus L_2(z)$, $c_i' \in C_i$ as c_i is only adjacent to vertices of C_i, and c_i' is not adjacent to any vertices of C_k. As $c_j \in C_j$, c_j is not adjacent to any vertex of C_k. Similarly, if $c_j' = P \setminus L_2(z)$, $c_j' \in C_j$ as c_j is only adjacent to vertices of C_j, and c_j' is not adjacent to any vertices of C_k. Thus, either c_i' and c_j' have a common neighbour in $N(z) = L_1(z)$ which is not adjacent to c_k (which is on $L_3(z)$), or P uses z itself (it cannot use a vertex of C_k as otherwise using z would use fewer vertices of C_k), which is also not adjacent to c_k.

Therefore, $\{c_i, c_j, c_k\}$ is as asteroidal triple, contradicting the fact that G is AT-free. □

Lemma 6.2.8 (Lemma 3.5, Gorzny and Huang [69]). *Let G be a connected bipartite permutation graph and v be an admissible vertex with $ecc(v) = diam(G) - 1$. Suppose that x, y are a diametrical dominating pair. Then v is adjacent to one of x, y. Moreover, there is a shortest (x, y)-path containing v.*

The LBFS end-vertex characterization on bipartite permutation graphs obtained by Gorzny and Huang [69] follows. It implies a polynomial time solution to LBFS-End-Vertex for bipartite permutation graphs: by testing all candidates for the vertex w, one can determine whether v is an LBFS end-vertex of G. However, this is not a linear time solution.

Theorem 6.2.9 (Theorem 3.6, Gorzny and Huang [69]). *Let G be a connected bipartite permutation graph and v be a vertex of G. Then v is an end-vertex of an LBFS if and only if there exists a vertex w such that, for every eccentric vertex u of w, $N(u) \subseteq N(v)$.*

6.3 The BFS-End-Vertex Problem for Bipartite Permutation Graphs

In this section, we show that the BFS-End-Vertex is in P for bipartite permutation graphs.

First, observe that since BFS is a layer search, the following observation holds.

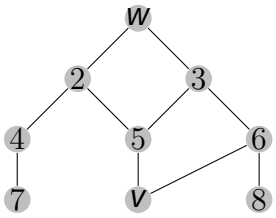


Figure 6.3: A vertex v of a bipartite permutation graph which is not a BFS end-vertex. The vertex v is an eccentric vertex of w.

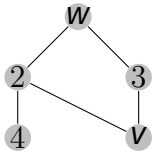


Figure 6.4: A vertex v of a bipartite permutation graph which is a BFS end-vertex but not an LBFS end-vertex.

Observation 6.3.1. *If v is a BFS end-vertex, then $d(v; w) = ecc(w)$ for some vertex w.*

Recall that in such a case, the vertex v is an *eccentric* vertex of w. In particular, if v is a BFS end-vertex of some BFS ordering , then $d(v;\ (1)) = ecc(\ (1))$.

It is tempting to think that any vertex v of a bipartite permutation graph which is an eccentric vertex can be a BFS end-vertex. Figure 6.3 shows a counter-example. To be last, BFS must start at a vertex where v is the last layer, in this case, that must be the vertex w. However, to make v last, 8 and 7 must come before it. But if BFS attempts to put 7 before v by choosing 2 after 1, then since 5 is a neighbour of 2, it must be placed into the queue before 6. Therefore, all neighbours of 5 on layer three will be placed before the neighbour of 6 on layer three, i.e., v will be before 8. Similarly, if 8 comes before v, then 7 will come after it. In that particular example, v is also not an LBFS end-vertex.

It may therefore also be tempting to think that every vertex which is a BFS end-vertex is also an LBFS end-vertex. However, Figure 6.4 shows a vertex which is a BFS end-vertex but not an LBFS end-vertex. To be last, LBFS must start at a vertex where v is the last layer, in this case, that must be the vertex w. Then, no matter how the vertices in layer one are chosen, v will always have a lexicographically larger label than the vertex 4.

Before characterizing BFS end-vertices on bipartite permutation graphs formally (in Theorem 6.3.7), we provide some intution. First, we will show that if the end-vertex has a start vertex which is very close, the graph behaves like a split graph, and we can use

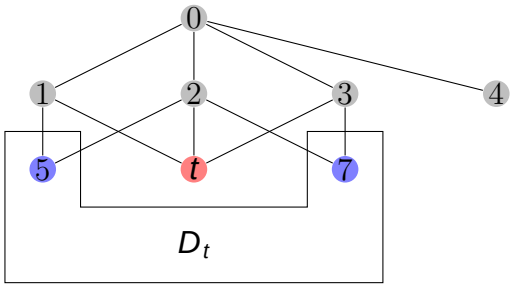


Figure 6.5: An illustration of the set D_t for a vertex t. In this example, $D_t = \{5, 7\}$.

the characterization of BFS end-vertices on split graphs. When the start vertex must be farther away from the end-vertex, we will use the structure of bipartite permutation to guide our characterization. From Lemma 6.2.7, we know that if we remove the start vertex for our desired ordering, we have at most two deep components. If we have one deep component (where our target end-vertex is in the last layer), Lemma 6.2.4 will be used to show that any vertex in the last layer is an end-vertex. If we have two, we will characterize a set of neighbours of our start vertex, X. The set X will need to separate the start vertex from the vertices in the last layer of the component that does not contain the target end-vertex. If all vertices of the last layer—including those in the same component as the target end-vertex—are not further away from X than the end-vertex itself, we will be able to end BFS at the target vertex.

The proofs only rely on the fact that Lemmas 6.2.4 and 6.2.7 hold. Both follow from the fact that bipartite permutation graphs are AT-free. It may be that for other graphs where similar lemmas hold, the same BFS end-vertex characterization also holds.

We start by handling some easy cases for when v is a BFS end-vertex. First, we observe that a necessary condition for BFS end-vertices shown by Charbit et al. [23] is sometimes also sufficient for bipartite permutation graphs. Second, we show that if $G \setminus N[w]$ has a single deep component for some w, then any vertex in the $L_{\mathrm{ecc}(w)}(w)$ can end a BFS starting at w.

We require the following definition from Charbit et al. [23]: for a vertex $t \in V(G)$, let $D_t = \{y \mid N_G(y) \setminus N_G(t)\}$. This definition[2] is illustrated in Figure 6.5.

The following condition was used by Charbit et al. [23] to characterize BFS end-vertices of split graphs.

[2]Charbit et al. [23] uses the term \dominates" to refer to a vertex which has a strictly larger neighbourhood than another. We do not use this term to avoid confusion with vertices in a \dominating pair".

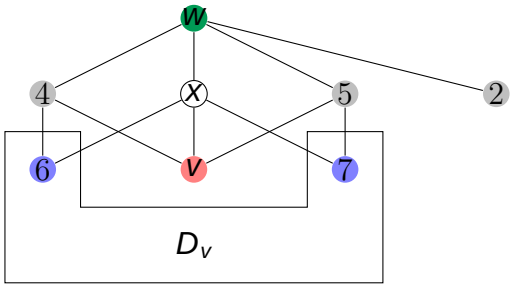


Figure 6.6: An example for the proof of Lemma 6.3.3. The target end vertex is v (in red), with the starting vertex w (in green). In this example, $D_v = \{6, 7\}$ (in blue), and the set S used in the proof is $S = \{2\}$. The special vertex $x \in N(v)$ (in white) is such that $D_v \subseteq N(x)$. One resulting BFS ordering is $= \langle w, 2, x, 4, 5, 6, 7, v \rangle$.

Theorem 6.3.2 (Theorem 3.14, Charbit et al. [23]). *Let $G = (V, E)$ be a graph, and let $t \in V$ be any vertex. A necessary condition for t to be the last vertex of some BFS ordering of G is that there exists a neighbour x of t such that $D_t \subseteq N(x)$.*

We are now ready to show that the condition of Theorem 6.3.2 is also sufficient when the start vertex is not too eccentric. More precisely, it is sufficient when the starting vertex w is such that $\text{ecc}(w) \leq 2$. An example graph for the proof is in Figure 6.6. For this proof, we do not require that the provided bipartite graph is also a permutation graph. The condition for split graphs works for these bipartite graphs since they have a similar structure: making $L_1(w)$ a complete graph for a bipartite graph where $\text{ecc}(w) = 2$ results in a split graph.

Lemma 6.3.3. *Let $G = (V, E)$ be a bipartite graph, and $v \in V$. Let $w \in V$ be such that $d(w, v) = \text{ecc}(w)$ and $\text{ecc}(w) \geq 2$. If there exists a neighbour x of v such that $D_v \subseteq N(x)$, then there is a BFS starting at w and ending at v.*

Proof. We will construct a valid BFS ordering ending at v. By assumption there is a vertex w such that $d(w, v) = \text{ecc}(w)$, $v \in L_{\text{ecc}(w)}(w)$, and $d(w, v) \geq 2$.

First, suppose that $d(w, v) \geq 1$. Since $v \in L_{\text{ecc}(w)}(w)$, $L_{\text{ecc}(w)}(w)$ is an independent set (by Observation 6.2.3), and $L_0(w) = \{w\}$, G must be a star centered at w. In this case, $x = w$ and $D_v = \emptyset$. After choosing w for the first vertex of the BFS ordering, all vertices of $V(G) \setminus \{w\} = N_G(w)$ are inserted into the queue: pick any insertion ordering that places v last.

Therefore, we assume that $d(w;v) = 2$. Since $x \in N(v)$, $d(v;w) = 2$, and $L_2(w) = L_{ecc(w)}(w)$ is an independent set (by Observation 6.2.3), we must have that $x \in L_1(w)$. Therefore, $(x;w) \in E(G)$.

Start BFS at w. Then, all neighbours of w are tied. Let $S \subseteq N(w)$ be the neighbours of w which are not adjacent to v, that is, $S = \{u \in N(w) | x \notin N(u)\}$. Place $N(w)$ in the queue such that all vertices of S appear first in any order, followed by x, and then any remaining vertices of $N(w)$ in any order. Let $S' \subseteq L_2(w)$ be the vertices of $L_2(w)$ with a neighbour in S, that is, $S' = \{u \in L_2(w) | N(u) \setminus S \neq \emptyset\}$. When visiting vertices of $L_2(w)$, BFS will necessarily choose all vertices of S' first.

Now, suppose $u \in L_2(w) \setminus (S' \cup \{v\})$. We claim that we can visit u before v. Any such vertex u has no neighbour in S, which means that $N(u) \subseteq N(v)$. If $N(u) \subset N(v)$, then $u \in D_v$ by choice of x; otherwise, $N(u) = N(v)$. In either case, $u \in N(x)$. Therefore, we can place all unvisited vertices of $N(x) \setminus \{v\}$ before v.

Finally, v remains to be visited, and BFS must now choose it. □

We now show that if v is an eccentric vertex of some other vertex w where $G \setminus N[w]$ contains a single deep component, then v is a BFS end-vertex. We first show the following, more general, helper lemma.

Lemma 6.3.4. *Let $G = (V;E)$ be a bipartite permutation graph. Suppose that there is a vertex w such that $ecc(w) \geq 3$ and $G \setminus N[w]$ has a deep component C. If there is a vertex $c \in C \setminus L_2(w)$ such that c is the first vertex of C visited by a BFS ordering and $(L_3(w) \setminus C) \subseteq N(c)$, then any vertex $v \in (C \setminus L_{ecc(w)}(w))$ can be the last vertex of C visited by BFS.*

Proof. Let h be the maximum value of $d(v;w)$ over all $v \in C$; clearly $h \leq ecc(w)$ and $h \geq 3$.

We will show that the BFS can be extended so that every permutation of $L_h(w) \setminus C$ can be the last $|L_h(w) \setminus C|$ vertices of C in the resulting ordering. The proof is by induction on h.

By Lemma 6.2.5, for each $2 \leq i \leq h-1$, $L_i(w) \setminus C$ has a vertex c_i such that $L_{i+1}(w) \subseteq N(c_i)$.

After visiting c_2, BFS must put $N_3^w = L_3(w) \setminus C$ into the queue to be visited later. BFS can insert these vertices in any ordering, and they will not be visited until $L_2(w)$ has been entirely visited. Inserting the vertices into the queue such that any vertex is last yields the result.

Therefore, the base case holds. Suppose there is an ordering for all $h = \ell \quad 3$, and consider $h = \ell + 1$.

By induction, there is a BFS ordering of $G' = G \quad (L_{\ell+1}(w) \setminus C)$ where the last $|L_\ell(w) \setminus C|$ vertices are in $L_\ell(w) \setminus C$; moreover, the vertices of $L_\ell(w) \setminus C$ can be arranged in any order. Apply the induction hypothesis to get any ordering $\quad$ of G' where the last $|L_\ell(w) \setminus C|$ vertices are c_ℓ (which exists by Lemma 6.2.5) followed by the remaining vertices of $L_\ell(w) \setminus C$. Therefore, after visiting c_ℓ, all of $L_{\ell+1}(w) \setminus C$ are inserted into the queue in whichever ordering is desired. Thus any vertex of $L_{\ell+1} \setminus C$ can be the last vertex visited of C. □

Corollary 6.3.5. *Let $G = (V, E)$ be a bipartite permutation graph, and $v \in V$. If there is a vertex w such that $d(v, w) = ecc(w)$, $ecc(w) \quad 3$, and $G \quad N[w]$ has a single deep component, then there is a BFS starting at w and ending at v.*

Proof. By Lemma 6.2.5, there is a vertex $c_2 \in L_2(w) \setminus C$ such that $(L_3(w) \setminus C) \quad N(c_2)$. Since G is connected, there is a vertex $w' \in N(w) = L_1(w)$ such that $c_2 \in N(w')$.

We construct $\quad$ as follows: after visiting w, place w' into the queue first, and then the rest of $N(w)$ in any order. Now the vertices of $L_2(w)$ can be visited so that c_2 is first. The result now follows from Lemma 6.3.4. □

The following fact regarding BFS is helpful.

Lemma 6.3.6. *Let $\quad$ be a BFS ordering. Suppose that $d(a, \quad) = k$, $d(b, \quad) = k$, and $d(a, \quad) > d(b, \quad)$ for some distinct vertices $a, b, \quad$, $\quad$ and non-negative integer k. If $a < b$, a is the leftmost vertex of $\quad$ such that $d(a, \quad) \quad k$, and b is the leftmost vertex of $\quad$ such that $d(b, \quad) \quad k$, then $\quad < \quad$.*

Proof. The proof is by induction on k. In the base case, $k = 1$. Since a is the leftmost vertex of $\quad$ such that $d(a, \quad) \quad k$ and b is the leftmost vertex of $\quad$ such that $d(b, \quad) \quad k$, neither $\quad$ nor $\quad$ have been visited when a is visited. Since $a < b$, $N(a)$ is inserted into the queue before $N(b)$. As $d(a, \quad) > d(b, \quad) = 1$, $\quad \in N(a)$ and therefore $\quad$ must have been visited before $\quad$ by BFS, i.e., $\quad < \quad$.

Suppose the result holds for all $k' \quad 1$ and consider the case $k = k' + 1$. Again, since a is the leftmost vertex of $\quad$ such that $d(a, \quad) \quad k$ and b is the leftmost vertex of $\quad$ such that $d(b, \quad) \quad k$, neither $\quad$ nor $\quad$ have been visited when a is visited.

Let $\quad'$ be the leftmost vertex in $\quad$ of a shortest $(a, \quad)$-path such that $d(a, \quad') = k \quad 1$. Similarly, let $\quad'$ be the leftmost vertex in $\quad$ of a shortest $(b, \quad)$-path such that $d(b, \quad') = k \quad 1$.

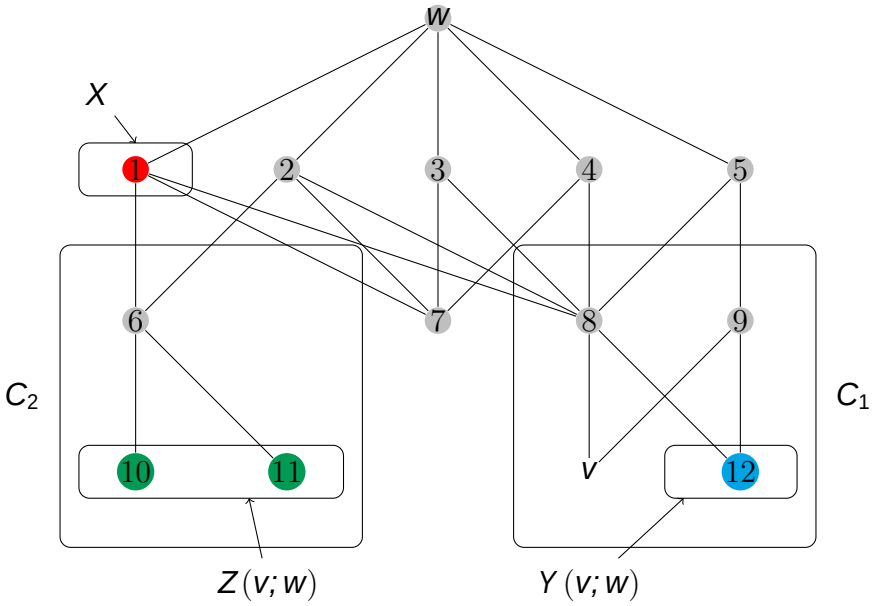


Figure 6.7: An example bipartite permutation graph with the sets used in Theorem 6.3.7 indicated. The deep components C_1 and C_2 of $G \quad N[w]$ are indicated. The vertex v is a BFS end-vertex: $= hw; 1; 2; 3; 4; 5; 6; 7; 8; 9; 10; 11; 12; vi$ is a BFS ordering ending at v. The green vertices are the set $Z(v;w)$, the blue vertices are the set $Y(v;w)$, and the red vertex is the necessary set $X(v;w)$.

By induction, $^{0} < \quad ^{0}$. Since $^{0} < \quad ^{0}$, $N(\ ^{0})$ is inserted into the queue before $N(\ ^{0})$. As $d(a; \ ) > d(b; \ ) = k$, $\ 2 N(\ ^{0})$ and therefore $\ $ must have been visited before $\ $ by BFS, i.e., $\ < \ $. □

We now prove the characterization of BFS end-vertices required for an efficient algorithm. Let v be a vertex such that $d(v;w) = \mathrm{ecc}(w)$ for some vertex w. Let $Z(v;w) = fu\ 2\ L_{\mathrm{ecc}(w)}(w)\ j\ N(v) \setminus N(u) = ;g$ and $Y(v;w) = fu\ 2\ L_{\mathrm{ecc}(w)}(w)\ n\ fvg\ j\ N(v) \setminus N(u)\ 6= ;g$. For a set $V^{0} \quad V(G)$ and vertex $x\ 2\ V(G)$, we denote $d(V^{0};x) = \min_{y2V^{0}} d(y;x)$. Figure 6.7 shows an example bipartite permutation graph with these sets indicated.

Theorem 6.3.7. *Let $G = (V;E)$ be a bipartite permutation graph and $v\ 2\ V$. Let $w\ 2\ V$ be such that $d(w;v) = \mathrm{ecc}(w)$ and $\mathrm{ecc}(w) \quad 3$. There is a BFS starting at w and ending at v if and only if, either*

- *$G \quad N[w]$ has a single deep component; or*

- *$G \setminus N[w]$ has two deep components C_1 and C_2, where without loss of generality $v \in C_1$, and there is a set $X \subseteq N(w) \setminus N(C_2)$ such that for each $u \in Y(v;w)$, $d(X;v) \geq d(X;u)$ and $d(X;u') = ecc(w) - 1$ for all $u' \in Z(v;w)$.*

Proof. We first show necessity. Let v be a BFS end-vertex for some BFS ordering starting with w. By assumption, $d(w;v) = \text{ecc}(w)$ and $\text{ecc}(w) \geq 3$; therefore $d(w;v) \geq 3$ and $G \setminus N[w]$ has at least one deep component.

If v is also an LBFS end-vertex, by Theorem 6.2.9, there must be an eccentric vertex u of w such that $N(u) \supseteq N(v)$; in such a case, every vertex of $L_{\text{ecc}(w)}(w)$ has a common neighbour with v, and therefore $G \setminus N[w]$ must have exactly one deep component and we are done. Therefore, we may assume that v is not also an LBFS end-vertex. By Theorem 6.2.9, there must be an eccentric vertex u of w such that either $N(u) \supset N(v)$ or $N(u)$ and $N(v)$ are incomparable. If all eccentric vertices of w are such that $N(u)$ and $N(v)$ are comparable, then $N^w_{\text{ecc}(w)-1}(u) \setminus N^w_{\text{ecc}(w)-1}(v) \neq \emptyset$ for any such u. That is, u and v are in the same component, and since there are no other eccentric vertices of w, it must be the case that $G \setminus N[w]$ has a single deep component. If $G \setminus N[w]$ has a single deep component we are done, as we have proved the first statement.

We may therefore assume that there is a vertex u of $L_{\text{ecc}(w)}(w)$ such that $N(u)$ and $N(v)$ are incomparable; u and v must be in different components of $G \setminus N[w]$ as otherwise we have a contradiction to Lemma 6.2.4. By Lemma 6.2.6, there are at most two deep components of $G \setminus N[w]$. Let C_1 and C_2 be the deep components of $G \setminus N[w]$; without loss of generality, assume that $v \in C_1$. In this case, $u \in C_2$ and by definition of $Z(v;w)$, $u \in Z(v;w)$ and thus $|Z(v;w)| \geq 1$.

Necessarily, $N(C_2)$ intersects every shortest $(u';w)$-path for every $u' \in Z(v;w)$. Therefore, $d(N(C_2);u) = \text{ecc}(w) - 1$ for all $u' \in Z(v;w)$ (this follows from the remark after Lemma 6.2.5). If $d(N(C_2);v) \geq d(N(C_2);v')$ for all $v' \in Y(v;w)$, then we may take $X = N(C_2)$ and there is nothing left to prove. In particular, if there is any subset $X \subseteq N(C_2)$ that hits every shortest $(u';w)$-path for every $u' \in Z(v;w)$ but no shortest $(v;w)$-path, then we are done.

We may therefore assume that every set $X \subseteq N(C_2)$ that intersects every shortest $(u';w)$-path for every $u' \in Z(v;w)$ also intersects at least one shortest $(v;w)$-path. Thus, we have that $d(X;v) = \text{ecc}(w) - 1$ for any such set X.

Among all possible choices for X such that $X \subseteq N(C_2)$ where X intersects every shortest $(u';w)$-path for every $u' \in Z(v;w)$, choose X to be as large as possible.

Claim 6.3.8. *For each $v' \in Y(v;w)$, $d(X;v') \geq d(X;v) = ecc(w) - 1$.*

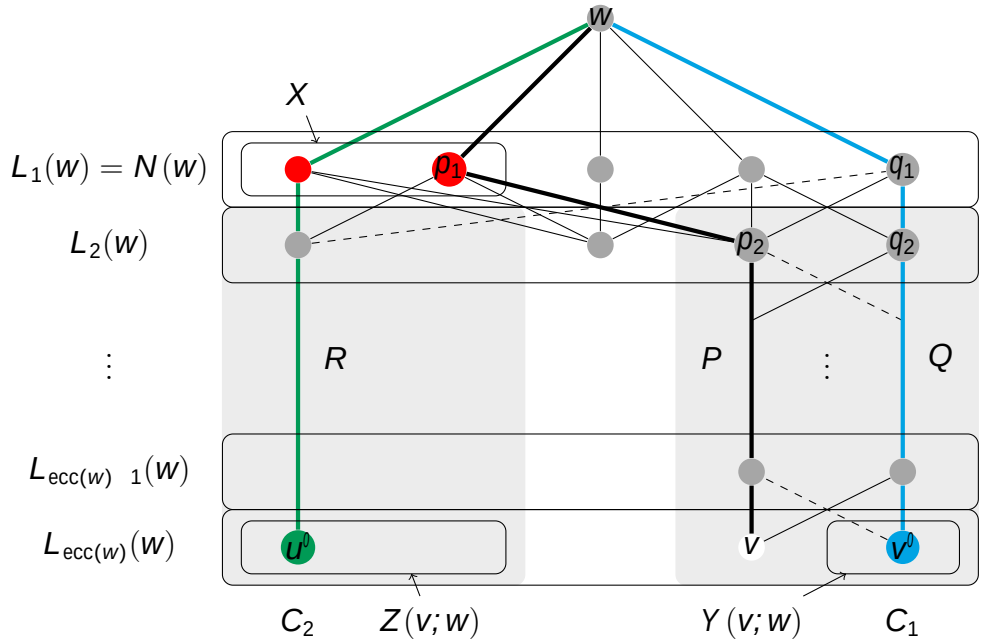


Figure 6.8: An illustration of the proof of Claim 6.3.8. Each layer may have more vertices than is shown. Regardless of other vertices, $u^0 \ 2 \ Z(v;w)$ is (one) vertex such that every set $X \quad N(C_2)$ that separates u^0 and w must also intersect a shortest $(v;w)$-path. The paths P, Q, and R are not necessarily the only such paths, but are drawn to illustrate the important properties of each path. The graph is drawn so that the path P is always left of the path Q in C_1 to clearly illustrate that p_i is not adjacent to q_{i+1}. Similarly, there may be other choices for R, but in any case, q_1 is not adjacent to $R \setminus L_2(w)$.

Proof of claim: We prove the claim by contradiction. The proof of the claim is illustrated in Figure 6.8. Suppose to the contrary that $d(X;v) < d(X;v^0)$ for some $v^0 \ 2 \ Y(z;w)$. Therefore, no shortest $(v^0;w)$-path is intersected by X, and X does not separate w and v^0.

Consider a shortest $(v;w)$-path P such that $P \setminus X \ 6= \ ;$ (we were allowed to assume such a path exists by the argument just prior to the claim's statement). Let $p_i \ 2 \ L_i(w) \setminus P$; p_i is unique for $1 \quad i \quad ecc(w)$. Among all choices for P, pick P so that p_1 is as leftmost as possible in .

Similarly, let Q be a shortest $(v^0;w)$-path such that $Q \setminus X = \ ;$ at least one such path Q must exist since X does separate w and v^0. Let $q_i \ 2 \ L_i(w) \setminus Q$; q_i is unique for

1 i ecc(w). Among all choices for Q, pick Q so that q_1 is as leftmost as possible in .

Since X was chosen to be as large as possible, $q_1 \notin R \setminus L_2(w)$ for any shortest $(u'; w)$-path R for any $u' \in Z(v; w)$. (If $q_1 \in R \setminus L_2(w)$, then $q_1 \in N(C_2)$, and we would have put q_1 in X; in this case, $d(X; v') = d(X; v)$, which is contrary to our assumption that $d(X; v) < d(X; v')$).

It must be that P is such that $N(q_{i+1}) \setminus \{p_i\} = \emptyset$ for every 1 i ecc(w) 1, as otherwise X intersects a shortest $(v'; w)$-path (take P but choose q_{i+1} instead of p_{i+1} after p_i for some i), which is a contradiction to X intersecting no such path. By Lemma 6.2.4, it must therefore be the case that $(N(q_i) \setminus L_{i-1}(w)) \subseteq (N(p_i) \setminus L_{i-1}(w))$ for every 2 i ecc(w). In particular, we must have that $N(v') \subseteq N(v)$.

We proceed by cases, based on whether $q_1 < p_1$.

Case 1: $q_1 < p_1$. Since $N(v') \subseteq N(v)$, $d(q_1; v) =$ ecc(w) 1. Since $q_1 \notin R \setminus L_2(w)$ for any shortest $(u'; w)$-path R (recall that $u' \in Z(v; w)$) and $q_1 \in N(w)$, $d(u'; q_1) =$ ecc(w) + 1 ($L_1(w)$ is an independent set by Observation 6.2.3, but there is a path of this length through w). By Lemma 6.3.6, since $d(q_1; v) =$ ecc(w) 1, $d(p_1; u') =$ ecc(w) 1, $d(q_1; u') =$ ecc(w) + 1 $> d(p_1; u') =$ ecc(w) 1, $v < u'$, a contradiction to the fact that v is the last vertex of . Thus the claim holds when $q_1 < p_1$.

Case 2: $p_1 < q_1$. By Lemma 6.3.6, since $d(q_1; v') =$ ecc(w) 1, $d(p_1; v) =$ ecc(w) 1, $d(p_1; v') =$ ecc(w) + 1 $> d(q_1; v) =$ ecc(w) 1, $v < v'$, a contradiction to the fact that v is the last vertex of . Thus the claim holds when $p_1 < q_1$.

We have therefore shown the claim.

Necessity follows from the claim, as all other cases have been covered prior to it.

For sufficiency, suppose that ecc(w) 3, $d(w; v) =$ ecc(w), and that $G N[w]$ has at least one deep component; thus $d(w; v)$ 3. If $G N[w]$ has a single deep component, we are done by Corollary 6.3.5. Otherwise, $G N[w]$ has two deep components by Lemma 6.2.6. Assume without loss of generality that $v \in C_1$. There are two cases:

Case 1: X does not intersect any shortest $(v; w)$-path. Let $w' \in N(w)$ be any neighbour of $c \in C_1 \setminus L_2(w)$ where $N(c) \setminus L_3(w) = L_3(w) \setminus C_1$ (which must exist by Lemma 6.2.4). Since $w' \in N(c)$, and c is on a shortest $(v; w)$-path, so is w'. Thus, $w' \notin X$. We can start BFS at w, then insert $N_G(w)$ into the queue as follows: $X; w'; N(w) \cap (X \cup \{w'\})$ (where X and $N(w) \cap (X \cup \{w'\}$ are inserted in any order). Then $L_{\text{ecc}(w)}(w) \setminus C_2$ will be visited before any vertices in $L_{\text{ecc}(w)}(w) \setminus C_1$, and those in $L_{\text{ecc}(w)}(w) \setminus C_1$ can be chosen so that v is last by Lemma 6.3.4. Thus v is a BFS end-vertex if X does not intersect any shortest $(v; w)$-path.

Case 2: X intersects some shortest $(v;w)$-path P. It must intersect a shortest path to w for every vertex in $Y(v;w)$ as well. To see this, let $v' \in Y(v;w)$ be arbitrary. If $N^w_{ecc(w)\ 1}(v) \neq N^w_{ecc(w)\ 1}(v')$, then we can take a shortest $(w;v)$-path that is intersected by X up to $L_{ecc(w)\ 1}$ and then append v' to get such a path. Otherwise, $N^w_{ecc(w)\ 1}(v') = N^w_{ecc(w)\ 1}(v)$ and $|N^w_{ecc(w)\ 1}(v) \cap N^w_{ecc(w)\ 1}(v')| \geq 1$. Let $v \in N^w_{ecc(w)\ 1}(v) \cap N^w_{ecc(w)\ 1}(v')$ be on a shortest $(v;w)$-path intersected by X (this must exist or every shortest $(v;w)$-path intersected by X also intersects a shortest $(v';w)$-path). Let $v^y \in N^w_{ecc(w)\ 1}$ be arbitrary. Thus we have that $N^w_{ecc(w)}(v^y) \neq N^w_{ecc(w)}(v)$. By Lemma 6.2.4 statement (4), we have that $N^w_{ecc(w)\ 2}(v) \neq N^w_{ecc(w)\ 2}(v^y)$. Thus we can take shortest $(v;w)$-path intersected by X up to a vertex of $N^w_{ecc(w)\ 2}(v)$, then take v^y and w to obtain a shortest $(v';w)$-path.

Partition X into disjoint sets $X_1 \cup X_2 \cup X_3$ such that X_1 consists of vertices which only intersect shortest paths from w to $Z(v;w)$, X_2 consists of vertices that only intersect shortest paths from w to $Z(v;w) \cup Y(v;w)$, and $X_3 = X \cap (X_1 \cup X_2)$. Note that X_3 is the set that contains vertices which are on some shortest path to v. Start BFS at w. Then, insert $N_G(w)$ into the queue as follows: $X_1; X_2; X_3$, where the vertices within X_1, X_2, and X_3 are ordered arbitrarily.

Starting with this prefix, we claim that BFS can order $L_{ecc(w)}(w)$ such that v is last. Any vertex $v' \in Y(v;w)$ must have $d(X;v') = ecc(w) - 1$ since $d(X;v) = ecc(w) - 1$ and by definition of X, $d(X;v) \leq d(X;v')$. If $d(X;v')$ is minimum due to a vertex $x \in X_2$, it will be visited before v as $x < x'$ for any $x' \in X_3$. Otherwise, $d(X;v')$ is minimum due a vertex $x' \in X_3$. Consider any shortest $(v;w)$-path P that contains x' and a shortest $(v';w)$-path P' which also contains x'. There must be a largest index $2 \leq i \leq ecc(w) - 1$ where $(P \setminus L_{i+1} \setminus C_1) \neq (P' \setminus L_{i+1} \setminus C_1)$ since these paths end at different vertices. At such an index i, we can place $(P' \setminus L_{i+1} \setminus C_1)$ into the queue before $(P \setminus L_{i+1} \setminus C_1)$, which will force v' to appear before v in the resulting ordering. Thus v is a BFS end-vertex if X does intersect some shortest $(v;w)$-path.

In either case, we have shown that v is a BFS end-vertex. □

The previous characterizations enable a proof of the next theorem, which is the main result for this section.

Theorem 6.3.9. *The* BFS-End-Vertex *problem for bipartite permutation graphs is in* P*.*

Proof. Let $G = (V;E)$ be a bipartite permutation graph. Algorithm 6.3.1 shows how to determine determine if $v \in V$ is a BFS end-vertex for G. We show its correctness, and that every step can be completed in polynomial time.

Algorithm 6.3.1: Polynomial Time BFS-End-Vertex solution for Bipartite Permutation Graphs.

Input: A bipartite permutation graph $G = (V; E)$ with vertex $v \in V$.

Output: `true` if there is a BFS ordering of G ending at v, or `false` otherwise.

```
1  S ← ∅;  X ← ∅;
2  for w ∈ V \ {v} do
3      if d(w, v) = ecc(w) then S ← S ∪ {w};
4  if |S| = 0 then return false;
5  for w ∈ S do
6      if d(w, v) ≠ 2 then
7          if ∃x ∈ N(v) such that D_v ≠ N(x) then return true;
8  for w ∈ S do
9      compute C_1 and C_2 of G \ N[w]; // assume that v ∈ C_1.
10     compute Y(v, w) and Z(v, w);
11     for w' ∈ N(w) \ N(C_2) do
12         if d(w', v) > ecc(w) − 1 then X ← X ∪ {w'};
13         if d(w', v) = ecc(w) − 1 then
14             if d(Z(v, w), w') ≠ ecc(w) − 1 and d(Y(v, w), w') ≠ ecc(w) − 1 then
15                 X ← X ∪ {w'};
16 if |X| = 0 then return false;
17 for u ∈ Z(v, w) do
18     if d(X, u) > ecc(w) − 1 then return false;
19 for v' ∈ Y(v, w) do
20     if d(X, v') > ecc(w) − 1 then return false;
21 return true;
```

Line 1 initializes some relevant sets; S will be the set of possible start vertices and X will be the set required by Theorem 6.3.7 if there is no start vertex w such that $d(w;v)$ 2.

Lines 2-3 compute potential start vertices w in polynomial time, by iterating over every w 2 $V(G)nfvg$ and recording those for which $d(w;v) = \text{ecc}(w)$. Distance can be computed in polynomial time by using, e.g., BFS. If there are none, we can immediately conclude that v is not a BFS end-vertex (line 4).

Lines 5-7 handle the case where BFS may need to begin at a vertex at distance at most two from v. If there are any potential start vertices w such that $d(w;v)$ 2 (which can be checked in polynomial time), we can check if the condition of Theorem 6.3.2 holds in polynomial time, which is also sufficient by Lemma 6.3.3. If any such w meets the requirements of Theorem 6.3.2, we can conclude that v is a BFS end-vertex (line 7).

Lines 8-15 handle the case where a start vertex must be distance at least three from v. For a possible start vertex w with $d(w;v)$ 3, we can also compute C_1, C_2 of G $N[w]$ in polynomial time. Without loss of generality, we can assume that v 2 C_1.

In polynomial time, we can see if w^0 2 $N(w) \setminus N(C_2)$ is on a shortest $(x;w)$-path for all x 2 $L_{\text{ecc}(w)}(w)$ by computing $d_{G\ fwg}(x;w^0)$ and comparing it to $d(w;v)$ 1. We can iterate over $W = N(w) \setminus N(C_2)$ (line 11) and add w^0 2 W to X if w^0 is not on any shortest $(v;w)$-path (line 12). If w^0 is on a shortest $(v;w)$-path, then we add w^0 2 W to X if and only if $d(w^0;u)$ ecc(w) 1 for all u 2 $Z(v;w)$ and $d(w^0;v^0)$ ecc(w) 1 for all v^0 2 $Y(v;w)$ (line 13). Since all of these checks, including computing the eccentricity of w, can be performed in polynomial time, the entire loop starting on line 8 runs in polynomial time too.

Line 16 handles the case where no X was able to be constructed. Note that if all possible start vertices are such that $d(w;v)$ 2 and none meet the required condition of Theorem 6.3.2, then X remains empty (as $N(w) \setminus N(C_2)$ is always empty on line 11) after executing lines 8-16. Thus, if we have reached line 16 and $jXj = 0$, we can conclude v is not a BFS end-vertex.

Lines 17-20 make sure that X satisfies the properties required of X by Theorem 6.3.7. Lines 13-15 make sure that every vertex in X satisfy the requirements, but lines 17-20 make sure that, e.g., no shortest $(u;w)$-path is missed for any u 2 $Z(v;w)$. Lines 17-20 only deal with distances between sets of vertices, and thus can be complete in polynomial time. That is, if jXj 1 and $d(X;u)$ ecc(w) 1 for all u 2 $Z(v;w)$ and $d(X;v^0)$ ecc(w) 1 for all v^0 2 $Y(v;w)$, then v is a BFS end-vertex; otherwise, v is not a BFS end-vertex. □

6.4 The LBFS-End-Vertex Problem for Bipartite Permutation Graphs

In this section, we show that LBFS-End-Vertex has a linear time solution on bipartite permutation graphs.

Let G be a bipartite permutation graph and let v be a vertex of G. Suppose that v is an end-vertex of LBFS of G. Then by Proposition 6.2.1 and Theorem 6.2.9, ecc(v) diam(G) 1 and there is a vertex w such that for every eccentric vertex u of w, $N(u)$ $N(v)$. When ecc(v) = diam(G), such a vertex w can be chosen to be any vertex with $d(w;v)$ = ecc(v) as shown by Gorzny and Huang [69]. The following theorem explains how to find such a vertex w in the case when ecc(v) = diam(G) 1.

The intuition for the theorem is as follows. We know from Lemma 6.2.7 that the bipartite permutation graph has at most two non-trivial components after removing a start vertex. From the characterization of LBFS end-vertices on bipartite permutation graphs (Theorem 6.2.9), we know that we want the target end-vertex to have the smallest neighbourhood among all eccentric vertices. If we happen to get the case where we have two non-trivial components after removing our start vertex, the theorem says how we can pick another start vertex, which will only have one deep component. It does this by prescribing a new start vertex which is closer to the eccentric vertices in the component that does not contain the target end-vertex than the eccentric vertices in the component containing the target end-vertex.

Theorem 6.4.1. *Let G be a connected bipartite permutation graph such that diam(G) = k. Suppose that v with ecc(v) k 1 is an LBFS end-vertex of G. If w is the rst vertex of $L_r(v)$ where $r = \frac{ecc(v)+3}{2}$ in an LBFS ordering of G that begins at v, then for every eccentric vertex u of w, $N(u)$ $N(v)$.*

Proof. Since v is an end-vertex of LBFS, there is an LBFS ordering of G with (n) = v (i.e., v is the last vertex in). Denote z = (1) and s = ecc(z). Clearly, v 2 $L_s(z)$.

The assumption that ecc(v) = k 1 implies immediately that s 2. As k 1 cannot be negative, s 1. If $s = 1$, then G is a star centered at z. If G is a star on two vertices, then diam(G) = 1 = ecc(v) = k, a contradiction to the fact that ecc(v) = k 1. However, if G is a star on more than two vertices, then diam(G) = 2 = ecc(v) = k, a contradiction to the fact that ecc(v) = k 1. Thus, s 2.

Claim 6.4.2. s 3.

Proof of claim: Assume to the contrary that $s = 2$. Then $N(v)$ $N(z)$ and 2 $\text{ecc}(v) =$ k 1 3.

Case 1: $\text{ecc}(v) = 2$. If $\text{ecc}(v) = 2$, then $N(v) = N(z)$ and hence for every eccentric vertex u of z, $N(u)$ $N(v)$. Since v is the last vertex in , we must have $N(u) = N(v)$ for every eccentric vertex u of z, which means G is a complete bigraph and $\text{ecc}(v) = 2 = k$, a contradiction to the assumption $\text{ecc}(v) = k$ 1.

Case 2: $\text{ecc}(v) = 3$. If $N(w) \setminus N(v) = ;$ for some eccentric vertex w of z, then $d(v;w)$ $4 > 3 = \text{ecc}(v)$, a contradiction. Therefore, it must be the case that $N(u) \setminus N(v) \neq ;$ for every eccentric vertex u of z. Since $k = \text{ecc}(v) + 1 = 4$, there are two eccentric vertices $u;w$ of z with $N(u) \setminus N(w) = ;$. Since v is an LBFS end-vertex, by Theorem 6.2.9, $N(w)$ $N(v)$ and $N(u)$ $N(v)$. Pick $w' \in N(w) \cap N(v)$ and $u' \in N(u) \cap N(v)$, and $(w';u';v)$ form an asteroidal triple in G, contradicting the assumption that G is AT-free.

Since we have a contradiction in both cases, we have $s = \text{ecc}(z)$ 3.

Let $C_1;C_2;:::;C_q$ be the connected components of G $N[z]$. Without loss of generality assume that $v \in C_q$. Let $x;y$ be a diametrical dominating pair in G which exists according to Theorem 6.2.2. By Lemma 6.2.8, v is adjacent to one of $x;y$. Assume by symmetry that v is adjacent to y. Then y is also in C_q and $d(z;y) = d(z;v)$ $1 = \text{ecc}(z)$ $1 = s$ 1.

Note that $d(x;y) = k = \text{ecc}(v) + 1$ $d(v;z) + 1 = s + 1$. Every vertex in $N[z] \cup C_q$ is at distance at most s from y and thus $x \notin N[z] \cup C_q$. Assume without loss of generality that $x \in C_1$. Denote $s_1 = d(x;z)$. The existence of an $(x;y)$-path of length $s_1 + s$ 1 (through the vertex z) implies that $k = d(x;y)$ $s_1 + s$ 1. On the other hand, any $(x;y)$-path through v is of length $s_1 + s$ 1. Hence, $k = d(x;y) = s_1 + s$ 1.

Consider a shortest $(x;y)$-path that contains v, which exists according to Lemma 6.2.8. Let P : $xx_1x_2:::x_k$ $_2vy$ be such a path. Then P contains a vertex in $N(z)$ as x and y belong to the different components of G $N[z]$. Let x $\in N(z)$ be the vertex in P with the smallest subscript and let Q be the subpath x x $_{+1}:::vy$ of P. Since

$$s = \text{ecc}(z) = d(z;v) \quad d(x ;v) + 1 = d(x ;y) \quad \text{ecc}(z) = s;$$

we have $d(x ;y) = s$, i.e., the length of Q is s. It follows that P does not contain z and moreover, if Q is replaced by an $(x ;y)$-path of length s through z then we obtain another shortest $(x;y)$-path P' containing z but not v. The existence of the shortest $(x;y)$-path P' (containing z) further implies every vertex in C_1 is at distance at most s_1 from z. Denote $P' : xx_1x_2:::x$ $zy_1y_2:::y_s$ $_2y$. Note that x $_1 \in N(x)\cap N(y_1)$ and x $\in N(x _{+1})\cap N(y_2)$.

Clearly, s_1 s. There are two cases, based on whether or not the inequality is strict.

Case A: $s_1 = s$. Then

$$r = \frac{\mathrm{ecc}(v)+3}{2} = \frac{s_1+s+1}{2} = \frac{2s+1}{2} = s+1.$$

Let be an LBFS ordering of G that begins at v. Observe that $\{x, y_1\} \quad L_{s\ 1}(v)$: by P, we know that $d(x, v) = s \quad 1$, and $d(y_1, v) = d(y, v_1) + 1 = (d(z, y) \quad 1) + 1 = ((s \quad 1) \quad 1) + 1) = s \quad 1$ as required by $P^{\emptyset}$. Observe further that $L_{s\ 1}(v) \quad N(z) \cup C_q$: every vertex in C_q must be hit by P, and therefore can be distance at most $s \quad 2$ away from z; in order for such a vertex to exist, it must be a neighbour of x , which means in any case that it is not in C_q.

It follows that the first vertex w of $L_r(v)$ in must be in $N(z)$. Since $d(w, v) = s+1$, w is not adjacent to x_{+1} (and w is not adjacent to x as they are both neighbours of z) and hence is adjacent to $x_{\ 1}$ as P is a dominating path. Now it is easy to see that the eccentric vertices u of w are all in C_q (as $d(w, x) = s_1 \quad 2 = s \quad 2 < s+1 = d(w, v)$) and satisfy the property that $N(u) \quad N(v)$.

Case B: $s_1 < s$. Then

$$r = \frac{\mathrm{ecc}(v)+3}{2} = \frac{s_1+s+1}{2} \quad \frac{2s}{2} = s.$$

In the case when $s_1 = s \quad 1$, w is a vertex adjacent to both x and y_1; note that $d(w, x) = s_1 = s \quad 1 < d(w, v) = s$. In the case when $s_1 \quad s \quad 2$, w is in C_q; note that $d(w, x)$ $s \quad 2 < s = d(w, v)$. In any case the eccentric vertices u of w are all in C_q and satisfy the property that $N(u) \quad N(v)$.

Thus the theorem is proved in either case. □

Algorithm 6.4.1 solves the LBFS-End-Vertex problem for bipartite permutation graphs. The correctness of the algorithm follows from Theorem 6.2.9, Theorem 6.4.1 and the remarks prior to Theorem 6.4.1. Theorem 6.2.9 establishes the conditions for a vertex to be an LBFS end-vertex. Then, Theorem 6.4.1 shows how to find the specific start vertex in the case the target end-vertex's eccentricity is not the diameter of the graph. Recall that LBFS can be implemented in linear time (e.g., Rose et al. [124]). Therefore a desired vertex w (as described in Theorem 6.2.9) can be found in linear time and verifying whether all eccentric vertices u of w satisfy the property that $N(u) \quad N(v)$ can also be done in linear time, the algorithm can be implemented to run in linear time. Therefore we have the following:

Theorem 6.4.3. *The* LBFS-End-Vertex *problem for bipartite permutation graphs can be solved in linear time.*

Algorithm 6.4.1: Linear Time LBFS-End-Vertex solution for Bipartite Permutation Graphs.

Input: A bipartite permutation graph G with vertex v.

Output: A vertex w such that $N(u) \quad N(v)$ for every eccentric vertex u of w in which case v is an end-vertex of G, or else v is not an end-vertex of LBFS of G.

1 Run LBFS beginning at v to find an LBFS ordering . Let $w_1 = (n)$ and w_2 be the first vertex of $L_r(v)$ in where $r = \frac{\mathrm{ecc}(v)+3}{2}$

2 Run LBFS beginning at w_1 to find all eccentric vertices of w_1. If $N(u) \quad N(v)$ for all eccentric vertices u of w_1, v is an end-vertex of LBFS of G.

3 Run LBFS beginning at w_2 to find all eccentric vertices of w_2. If $N(u) \quad N(v)$ for all eccentric vertices u of w_2, v is an end-vertex of G; otherwise v is not an end-vertex of LBFS of G.

6.5 The DFS-End-Vertex Problem for Bipartite Permutation Graphs

In this section we show that we can identify a DFS end-vertex in linear time on bipartite permutation graphs. To do this, we refine the following characterization of (arbitrary) DFS end-vertices of Kratsch et al. [92]. A *Hamiltonian* path P of a graph G is a path such that $V(P) = V(G)$ and all vertices of P are distinct.

Theorem 6.5.1 (Proposition 2, Kratsch et al. [92])**.** *Let G be a connected graph, and let t be a vertex of G. Then t is a DFS end-vertex of G if and only if there is $X \quad V(G)$ such that $N_G[t] \quad X$ and $G[X]$ has a Hamiltonian path with endpoint t.*

Our refinement allows us to use a strong ordering to find the required set X. We describe this informally before stating the condition in the next corollary. Suppose that G is a bipartite permutation graph G, (^{A}; ^{B}) is a strong ordering of G, and $v \; 2 \; A$ is the target DFS end-vertex. Because G is connected, ^{A} has both the adjacency and enclosure properties (Theorem 2.3.7). Therefore, $N(v) \quad B$ is consecutive in ^{B}. Then, if a set X as required by Theorem 6.5.1 exists, we only need to look at the vertices between the leftmost vertex of ^{A} which is adjacent to a vertex in $N(v)$ and the rightmost vertex of ^{A} which is adjacent to a vertex in $N(v)$ in order to find it. This is because a Hamiltonian path would not need to use any more vertices of B. An example is shown in Figure 6.9.

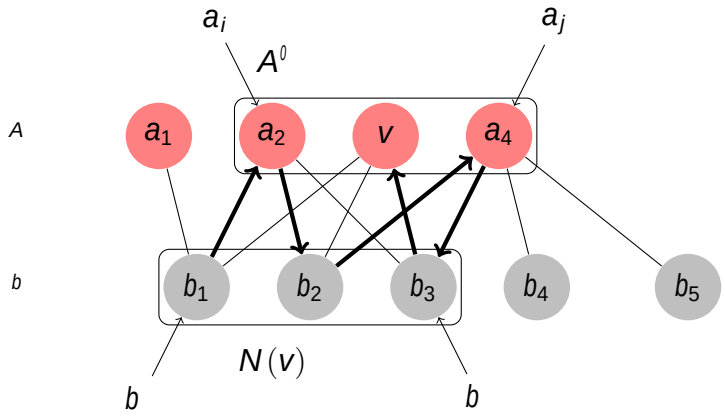


Figure 6.9: A vertex v of a bipartite permutation graph which is a DFS end-vertex. One partite set, A, is indicated in red on the top row, while the other, B, is in grey on the bottom row. Each partite set is drawn so that the resulting orderings form a strong ordering $(\ ^{A}; \ ^{B})$ $(v = a_3)$. The path $fb_1; a_2; b_2; a_4; b_3; vg$ is a Hamiltonian path for $G[A' \ [\ N(v)]$ and is indicated by thick arrows. A DFS ending at v is $= hb_1; a_1; a_2; b_2; a_4; b_4; b_5; b_3; vi$.

Theorem 6.5.2. *Let* $(\ ^{A}; \ ^{B})$ *be a strong ordering of a bipartite permutation graph. Without loss of generality, assume* v *2* A.

Vertex v *is a DFS end-vertex if and only if there is a consecutive set* $A' \quad ha_i; : : : ; a_j i$ *such that* $a_i \quad _A \ v \quad _A \ a_j$, $G' = G[A' \ [\ N(v)]$ *is an induced subgraph of* G *where* $jA'j = jN(v)j$ *and* G' *has a Hamiltonian path ending at* v.

Proof. Sufficiency is immediate from Theorem 6.5.1. We therefore only show necessity.

Suppose that v is a DFS-end vertex. Therefore, $v = a_i$ for some i.

By Theorem 6.5.1, since v is a DFS end-vertex, there must be a set X such that $N[v] \quad X$ and X induces a subgraph $G' = G[X]$ of G which has a Hamiltonian path ending at v. We will show it is not necessary to include any vertices of B other than $N(v) \quad B$.

If $d(v) = 1$, we are done by Theorem 6.5.1 and taking $A' = fvg$; clearly we do not need any vertices of $B \ n \ N(v)$. We may therefore assume that $d(v) \quad 2$.

Claim 6.5.3. $X \setminus B = N(v)$.

(a) The initial con guration of the Hamiltonian path H.

(b) The new con guration of the Hamiltonian path H.

Figure 6.10: Case 1 of the proof of Theorem 6.5.2 illustrated.

Proof of claim: Suppose to the contrary that for every X $V(G)$ that contains $N[v]$ has a Hamiltonian path ending at v, $X \setminus (B \cap N(v)) \neq ;$. Among all possible choices of X, choose one which minimizes $|X|$. Let $X' = X \cap N[v]$.

There are no vertices $x \in X'$ such that $d_{G'}(x) = 1$. Suppose to the contrary that there is such a vertex u. The vertex u must be the endpoint of the Hamiltonian path H which is not v. Moreover, since $u \in X'$, $u \notin N(v)$. Therefore, we could start the Hamiltonian path at the neighbour of u on H to get a smaller set X, contradicting ouR choice of a minimal set X.

Since $(\ ^A;\ ^B)$ is a strong ordering of G, $N(v)$ is consecutive in $\ ^B$ by Theorem 2.3.7. Without loss of generality, we may assume that there is a $b \in X'$ such that $b <_B N(v)$, as if this is not the case, we can use $((\ ^A)^R;(\ ^B)^R)$ as the initial strong ordering of G. Let $b \in X'$ be as leftmost as possible in $\ ^B$ as possible. Let H be a Hamiltonian path of $G[X]$ ending at v. If b is the first vertex of H, then clearly we can remove b from X, contradicting our choice of a minimal X. Let $a' \in A$ be the vertex immediately before b in H, and $a'' \in A$ be the vertex immediately after b in H.

We proceed by cases based on whether a'' is before or after a' in $\ ^A$. Case 1 is illustrated in Figure 6.10.

Case 1: $a'' <_A a'$. Since $b \notin N(v)$, $a'' \neq v$. There must is a vertex $b' \in X$ after a'' in H, and since b was earliest in $\ ^B$, $b <_B b'$ (recall also that $b <_B N(v)$). Since $(b;a') \in E$, $(a'';b') \in E$, and $(\ ^A;\ ^B)$ is a strong ordering, $b'a' \in E$. Since H is a Hamiltonian path, a' and b' are only visited once, in particular, the edge $(a';b')$ is never used. Therefore, we can remove $(a'';b)$ from H and replace it by the edge $(a';b')$ to get a shorter Hamiltonian path H'. Since $b \notin N(v)$, $N[v]$ H' still, and H' contains a Hamiltonian path ending at v, i.e., we could have taken H' instead of X, contradicting our choice, as $|H'| < |X|$ and we chose X so that $|X|$ was minimized.

Case 2: $a' <_A a''$. If a' is the first vertex of H, then we can remove both a' and b from H and start a new Hamiltonian path H' at a'', using an X which does not include a' or b, contradicting the minimality of X. Therefore, there is a vertex $b' \in B$ such that $b <_B b'$ where $(b'; a') \in E$. Since $(b; a'') \in E$, $(a'; b') \in E$, and $(<_A; <_B)$ is a strong ordering, $(a''; b') \in E$. Since H is a Hamiltonian path, a'' and b' are only visited once, in particular, the edge $(a''; b')$ is never used. Therefore, we can remove $(a'; b)$ from H and replace it by the edge $(a''; b')$ to get a shorter Hamiltonian path H'. Since $b \notin N(v)$, $N[v] \not\subseteq H'$ still, and H' contains a Hamiltonian path ending at v, i.e., we could have taken H' instead of X, contradicting our choice, as $|H'| < |X|$ and we chose X so that $|X|$ was minimized.

Since we have a contradiction in either case, the claim is proved.

It remains to be shown that $X \setminus A$ is consecutive, in which case we can partition X into $X \setminus A$ and $X \setminus B$, and take $A' = X \setminus A$ to complete the proof.

Claim 6.5.4. *There is a smallest X such that $X \setminus A$ is consecutive in $<_A$ where $X \setminus B = N(v)$.*

Proof of claim: Suppose to that for the set X, there are vertices $a_i <_A a_j <_A a_k$ where $a_i, a_k \in (X \setminus A)$ but $a_j \notin X$.

Necessarily, $v \in X$, so $a_j \neq v$. Without loss of generality, assume that $v \in (<_A)_i$, as otherwise we can reverse both $<_A$ and $<_B$. Therefore, we also have that $a_k \neq v$.

Since X is as small as possible and $X \setminus B = N(v)$, the Hamiltonian path H contained in X does not end at a vertex of $A \cap \{v\}$. Therefore, there are two vertices $b, b' \in H$ that are both adjacent to a_k. By relabelling these vertices, we may assume that $b <_B b'$. Since $X \setminus B = N(v)$, $b, b' \in N(v)$.

Since $<_B$ has the adjacency property, $v <_A a_k$, and $v, a_k \in N(b)$, it must be that $a_j \in N(b)$. Since $v <_A a_j$, $b <_B b'$, $(v, b'), (a_j, b) \in E(G)$ and $(<_A; <_B)$ is a strong ordering, we must have that $a_j \in N(b')$.

Thus, we could replace a_k by a_j in H. We can repeat this for all $v <_A a_{k'}$ where $a_{k'} \in X$. Thus, $(X \setminus A) \setminus (<_A)_v$ is consecutive. Symmetrically, we can ensure that $(X \setminus A) \setminus (<_A)_v$ is consecutive. The resulting set X is such that $X \setminus A$ is symmetric.

Since we have not changed any vertices of $X \setminus B$, we still have that $X \setminus B = N(v)$, and the proof of the claim is complete.

The previous claim completes the proof of the corollary. □

A Hamiltonian cycle C of a graph G is a cycle of length $|V(G)|$ such that $V(C) = V(G)$. If there is a set provided by the previous corollary and the end-vertex is not a pendant,

then we also have a Hamiltonian cycle in the graph induced by the set, as shown in the next proposition. This is helpful to simplify the complexity of finding such a set.

Proposition 6.5.5. *Let $G = (A; B; E)$ be bipartite permutation graph and let $v \in A$ be a DFS end-vertex such that $d(v) > 1$. Let $X = A' \cup N(v)$ be a minimal set such that $G[X]$ has a Hamiltonian path ending at v, $X \setminus B = N(v)$, and A' is a consecutive set of $\mathcal{A}$ containing v (i.e., X is a minimal set provided by Theorem 6.5.2).*

Then $G[X]$ has a Hamiltonian path if and only if $G[X]$ has a Hamiltonian cycle.

Proof. Suppose first that $G[X]$ has a Hamiltonian cycle $H = \{x_1; \ldots; x_{|X|} = v\}$. Since $(x_1; v) \in E$ and $v \in A$, we must have that $x_1 \in B$. Therefore, we can remove the edge $(x_1; v)$ from H to get a Hamiltonian path ending at v.

Suppose instead that $G[X]$ has a Hamiltonian path $H = \{x_1; \ldots; x_{|X|} = v\}$.

We claim that $|H| = 2|N(v)|$. Since G is bipartite, x_i must be in a different set from x_{i+1} for all $1 \leq i < |X|$. Since $v \in A$ is last and the sets of each x_i alternate, we must have that every $b \in N(v)$ is followed by a vertex in $A' \cap A$. However, if there is a vertex $a \in A$ which is before the first vertex of B in H, we can remove it to get a smaller set X. We must therefore have had that $|X \setminus A| = |X \setminus B|$. Since $X \setminus B = N(v)$, we have that $|X| = 2|N(v)|$.

Since $|H| = 2|N(v)|$, each vertex alternates partite sets, and H ends at $v \in A$, we must have that $x_1 \in B$. Since $X \setminus B = N(v)$, we have that $(x_1; v) \in E$ and we can add $(x_1; v)$ to H to get a Hamiltonian cycle containing v. □

Using the following condition for when a Hamiltonian path exists in bipartite permutation graphs, we can show the main result of the section.

Theorem 6.5.6 (Theorem 2, Brandstädt and Kratsch [17])**.** *If $G = (A; B; E)$ is a bipartite permutation graph with $|A| = |B|$, then G has a Hamiltonian cycle if and only if for each $1 \leq i < |A|$, $(a_i; b_i); (a_i; b_{i+1}); (a_{i+1}; b_i); (a_{i+1}; b_{i+1}) \in E$. Therefore, determining if G has a Hamiltonian cycle takes $O(n)$ time.*

Theorem 6.5.7. *The* DFS-End-Vertex *problem for bipartite permutation graphs can be solved in linear time.*

Proof. Algorithm 6.5.1 solves DFS-End-Vertex. We now show its correctness. Recall that a strong ordering $(\mathcal{A}; \mathcal{B})$ of a bipartite permutation graph $G = (A; B; E)$ can be

Algorithm 6.5.1: Linear Time DFS-End-Vertex solution for Bipartite Permutation Graphs.

Input: A bipartite permutation graph $G = (A; B; E)$, a strong ordering (^{A}; ^{B}) of G, and a vertex v 2 A.

Output: true if there is a DFS ordering of G ending at v, or false otherwise.

```
1  if d(v)    1 then return true ;
2  i'    i (where  ^A(v) = i);
3  r    0; j'      1 ;
4  if i' < jAj and 9b_j 2 N(v) such that (b_j; a_{i'+1}) 2 E then
5      j'    j;
6      while true do
7          if i' = jAj or i' > i + jN(v)j or j' > jBj or b_{j'} 2 N(v) then break ;
8          if (a_{i'}; b_{j'}); (a_{i'}; b_{j'+1}); (a_{i'+1}; b_{j'}); (a_{i'+1}; b_{j'+1}) 2 E then
9              r    r + 1; i'    i' + 1; j'    j' + 1;
10         else
11             break;
12 if r    jN(v)j    1 then return true;
13 i'    i (where  ^A(v) = i);
14 `    0; k'      1 ;
15 if 1 < i' and 9b_k 2 N(v) such that (b_j; a_{i'  1}) 2 E then
16     k'    k;
17     while true do
18         if i' = 1 or i' < i    jN(v)j or k' = 1 or b_{k'} 2 N(v) then break ;
19         if (a_{i'}; b_{k'}); (a_{i'}; b_{k'  1}); (a_{i'  1}; b_{k'}); (a_{k'  1}; b_{k'  1}) 2 E then
20             `    ` + 1; i'    i'    1; j'    j'    1;
21         else
22             break;
23 if `    jN(v)j    1 then return true;
24 if j'    k' and (j'    j + 1) + (k    k' + 1)    (j'    k' + 1)    jN(v)j then
25     return true;
26 else if j' < k' and (j'    j + 1) + (k    k' + 1) + (k'    j'    1)    jN(v)j then
27     return true;
28 return false;
```

computed in linear time (Chang et al. [22]). Without loss of generality, assume that $v \in A$ and therefore $N(v) \cap B$.

Line 1 handles the easiest cases where v has very small degree. Therefore, we may assume that $d(v) \geq 2$ after line 1.

Lines 2-22 count the number of vertices of A that could be used for the set X which is both necessary and sufficient by Theorem 6.5.2.

Lines 2-11 count the number of vertices of A that could be used for the set X which are to the right of a in $\prec_A$. Starting at $v = \prec_A(v)$, we will see if there are any vertices of $N(v)$ that have a neighbour that follows v in $\prec_A$ (the condition on line 4). Note that we can find a vertex b_j for line 4 in linear time by pre-processing the graph: we can scan each vertex of $\prec_B$ and record its rightmost neighbour in $\prec_A$ (this takes time $O(n+m)$). Then, we see if the vertex to the right of b_j (i.e., b_{j+1}) is adjacent to the vertex right of a_{i+1} (i.e., a_{i+2}), and continue until this this condition fails (lines 8-11). We may also stop if the indices of vertices we wish to consider are no longer valid or no such vertex exists (line 7). Once the condition fails, we break out of the loop (lines 10-11). We break out of the loop because we know, by Proposition 6.5.5 and Theorem 6.5.6, that the condition is required for a Hamiltonian path; if it is not met, we are wasting time.

In particular on line 12, if $r > 0$, there is a subgraph $G_r = (A_r; B_r; E_r)$ of G such that $|A_r| = |B_r| = r$, $A_r = \{v = a_i, \ldots, a_{i+r}\}$, $B_r \subseteq N(v)$ and G_r has a Hamiltonian cycle by Theorem 6.5.6. Then we know v is a DFS end-vertex by Theorem 6.5.2 if $r \geq |N(v)| - 1$.

Lines 13-22 count the number of vertices of A that could be used for the set X which are to the left of a in $\prec_A$. Starting at $v = \prec_A(v)$, we will see if there are any vertices of $N(v)$ that have a neighbour that is before v in $\prec_A$ (the condition on line 15). Note that we can find a vertex b_j for line 15 in linear time by pre-processing the graph: we can scan each vertex of $\prec_B$ and record its leftmost neighbour in $\prec_A$ (this takes time $O(n+m)$). Then, we see if the vertex to the left of b_j (i.e., b_{j-1}) is adjacent to the vertex left of a_{i-1} (i.e., a_{i-2}), and continue until this this condition fails (lines 19-22). We may also stop if the indices of vertices we wish to consider are no longer valid or no such vertex exists (line 18). Once the condition fails, we break out of the loop (lines 21-22). We break out of the loop because we know, by Proposition 6.5.5 and Theorem 6.5.6, that the condition is required for a Hamiltonian path; if it is not met, we are wasting time.

In particular on line 23, if $\ell > 0$, there is a subgraph $G_r = (A_r; B_r; E_r)$ of G such that $|A_r| = |B_r| = r$, $A_r = \{v = a_i, \ldots, a_{i+r}\}$, $B_r \subseteq N(v)$ and G_r has a Hamiltonian cycle by Theorem 6.5.6. Then we know v is a DFS end-vertex by Theorem 6.5.2 if $\ell \geq |N(v)| - 1$.

From line 24 onward, we are considering two cases which imply that v is a DFS end-vertex (lines 24 and 26). Note that if we enter either case, we must have that $k' > k - 1$

and $j' > 1$ (i.e., the vertices b_j and b_k defined in lines 4 and 15 exist).

In the first case (line 24), we are ensuring that the number of vertices in B which are adjacent to a vertex to the right of v in A ($j' \ldots k+1$) and the number of vertices in B which are adjacent to a vertex to the left of v in B ($k \ldots k'+1$), minus the number of vertices which are adjacent to vertices on either side of v in A ($j' \ldots k'+1$) is at least $|N(v)|$. Let $s = \min\{\ell, |N(v)| - 1\}$ (and note that $|N(v)| - 1 \ge s \ge 0$ as $\ell \ge 0$ and $N(v) \ge 2$), and let $t = |N(v)| - 1 - s$ (note that $t \ge 0$). Consider the subgraph $G' = (A'; B'; E')$ of G where $A' = \langle a_{i-s}, \ldots, a_{i-1}, v = a_i, a_{i+1}, \ldots, a_{i+t} \rangle$, $B' = \langle b_j, \ldots, b_k \rangle$. Since $B' \subseteq N(v)$ and $(j' \ldots j+1) + (k \ldots k'+1) \ge (j' \ldots k'+1) \ge |N(v)|, |B'| \ge |N(v)|$. Then G' has a Hamiltonian cycle by Theorem 6.5.6 and the conditions enforced on lines 8 and 19. Therefore v is a DFS end-vertex by Theorem 6.5.2.

In the second case (line 24), we are ensuring that the number of vertices in B which are adjacent to a vertex to the right of v in A ($j' \ldots k+1$) and the number of vertices in B which adjacent to a vertex to the left of v in B ($k \ldots k'+1$), plus the number of vertices which are only adjacent to vertices on one side of v in A ($j' \ldots k'+1$) is at least $|N(v)|$. Let $s = \min\{\ell, |N(v)| - 1\}$ (and note that $|N(v)| - 1 \ge s \ge 0$ as $\ell \ge 0$ and $N(v) \ge 2$), and let $t = |N(v)| - 1 - s$ (note that $t \ge 0$). Consider the subgraph $G' = (A'; B'; E')$ of G where $A' = \langle a_{i-s}, \ldots, a_{i-1}, v = a_i, a_{i+1}, \ldots, a_{i+t} \rangle$, $B' = \langle b_j, \ldots, b_k \rangle$. Since $B' \subseteq N(v)$ and $(j' \ldots j+1) + (k \ldots k'+1) + (k' \ldots j' \ldots 1) \ge |N(v)|, |B'| \ge |N(v)|$. Then G' has a Hamiltonian cycle by Theorem 6.5.6 and the conditions enforced on lines 8 and 19. Therefore v is a DFS end-vertex by Theorem 6.5.2.

If we have not yet returned, then we know v is not a DFS end-vertex (line 28). □

6.6 The MNS-End-Vertex Problem for Bipartite Permutation Graphs

In this section, we show that MNS-End-Vertex problem for bipartite permutation graphs is in P.

Before formally characterizing MNS end-vertices on bipartite permutation graphs in Theorem 6.6.1, we provide some intuition. One way that a vertex v would be last in an MNS ordering is if we could visit all the components of $G \setminus N[v]$, along with $N(v)$ itself, before visiting v. We first prove that this is possible if there is a vertex in $N(v)$ which is adjacent to all the components of $G \setminus N[v]$. Then, starting MNS at that vertex, we can visit the components in a sort of depth-first manner (though not strictly depth-first).

For the trivial components, this is easy, but it turns out this is also possible for the at most two (Lemma 6.2.6) non-trivial components of $G \quad N[v]$ as well. Since MNS does not say *which* maximal label we should take, we can entirely visit a component before going to another. Then, all that remains is to show that this can be done so that the neighbours of v must also be picked before v itself. Moreover, it turns out that a vertex in $N(v)$ adjacent to all components is also necessary. The proof is more complicated, and proceeds by contradiction. The contradiction is reached when we show that if no such vertex adjacent to all components exists, then the graph should have been infinite. In particular, if there are two components which do not share a common neighbour in $N(v)$, then v should have been chosen between the components, unless the graph was infinitely large.

We use one definition from Berry et al. [6] for our characterization of MNS end-vertices on bipartite permutation graphs. We do this in order to make the differences in the characterizations apparent when comparing our results with theirs (e.g., their MNS end-vertex characterization for chordal graphs). Let $G = (V; E)$ be a graph and $v \; 2 \; V$. Let $C_1; : : : ; C_t$ be the connected components of $G \quad N[v]$. The *substars* of v are the elements of $N_G(C_i)$ for each i. See Figure 6.11 for an illustration of substars.

The MNS end-vertex characterization for bipartite permutation graphs follows; its proof is delayed until after some helpful lemmas are established. Figure 6.11 illustrates the condition. As a corollary, MNS-End-Vertex can be solved efficiently using standard techniques. Recall that a cut vertex is a vertex v such that $G \quad fvg$ has more components than G.

Theorem 6.6.1. *Let G be a connected bipartite permutation graph and let v be a vertex of G. Then v is an MNS end-vertex if and only if v is not a cut vertex and there is a vertex y 2 N(v) that is in every substar of v.*

Every MNS-generated ordering is also a generic search ordering (see Figure 2.19). Therefore, if a vertex is an MNS end-vertex, it is also a generic search end-vertex. The following theorem characterizes end-vertices of generic search, and is the reason we require that MNS end-vertices are not cut vertices.

Theorem 6.6.2 (Theorem 2.1, Charbit et al. [23])**.** *A vertex v is the end-vertex of some generic search of G if and only if it is not a cut vertex of G.*

We now show sufficiency. The following lemma shows that MNS can visit a component of $G \quad N[v]$ entirely once it has been started and simplifies the proof. Let $`(v)$ denote the label of a vertex assigned by MNS.

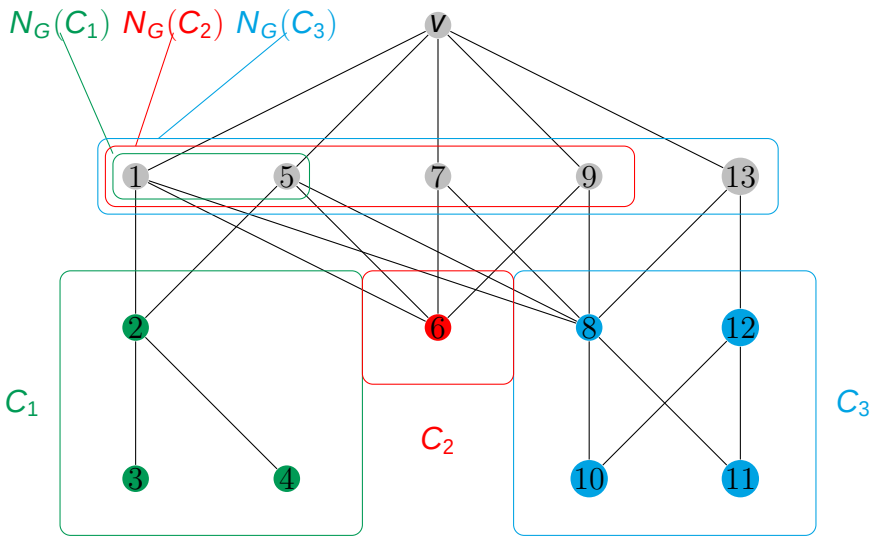


Figure 6.11: An example bipartite permutation graph G where the components C_1, C_2, and C_3 of $G \quad N[v]$ are indicated. For each component C_i $(1 \quad i \quad 3)$, $N_G(C_i)$ is the substar of C_i. The sets used in Theorem 6.6.1 are the substars of the components; vertex 5 is in all the substars of $G \quad N[v]$. The vertex v is a MNS end-vertex: $= h1; 2; 3; 4; 5; 6; 7; 8; 9; 10; 11; 12; 13; vi$ is an MNS ordering ending at v.

Lemma 6.6.3. *Let G be a connected bipartite permutation graph, and let C be a non-trivial component of $G \quad N[v]$ for some $v \; 2 \; V(G)$. Suppose that there is a vertex $y \; 2 \; N(v)$ that is in every substar of v. Finally, let $T \quad V$ be the trivial components of $G \quad N[v]$.*

If the vertices of $fyg \; [\; T$ are the only vertices numbered by MNS and y was numbered rst, then $C \; [\; (N(C) \; n \; fyg)$ can be the next $jCj + jN(C)j \quad 1$ vertices numbered by MNS.

Proof. Let q be a vertex in C as far away from v as possible, and let $h^0 = d_G(v; q)$; note that $h^0 \quad 3$ as otherwise C is not a non-trivial component. For $2 \quad h \quad h^0$, let $c_h \; 2 \; L_h(v) \setminus C$ be such that $(L_{h \; 1}(v) \setminus N[C]) \quad N(c_h)$; that is, c_h is adjacent to all vertices of $N[C]$ on the layer above c_h in $N[C]$. Such a vertex always exists by Lemma 6.2.5.

The vertex c_2 can be chosen next by MNS, since $y \; 2 \; N(v) \setminus N(C)$ and $(N(c_2) \setminus N(v)) = N(C)$.

We prove that the next $jCj \quad 1$ vertices chosen are those of $C \; n \; fc_2g$. The proof is by induction on h^0.

In the base case, suppose that $h' = 3$. After choosing c_2, each vertex in $N(C) \setminus \{c_3\}$ receives the number given to c_2 in its label, as does c_3. Since we have numbered the vertices of T, we may not be able to choose c_3 before some vertices of $N(C) \setminus \{c_3\}$; label as many vertices of $N(C) \setminus \{c_3\}$ as required by MNS before we can choose c_3. Since $c_3 \in L_3(v)$ and $(N(C) \setminus \{c_3\}) \cap L_1(v)$, the vertices on $L_3(v) \setminus C_2$ never have their label updated by visiting vertices of $N(C) \setminus \{c_3\}$. Therefore, we can choose c_3 next. Then, we can label the vertex c_2', where c_2' is such that $(L_3(v) \setminus C) \cap N(c_2')$ (which exists by Lemma 6.2.5), if it was not already labeled (it may be that $c_2' = c_2$). Following c_2', we can label by $L_2(v) \setminus C$ in any order: all such vertices have maximal labels with the number given to c_3. Then, all vertices of $L_3(v) \setminus C$ can be numbered in some order, as all of their neighbours have been visited.

Therefore, assume that the result holds for $h' \geq 3$ and consider $h'+1$. By induction, all vertices of C at distance at most h' from v can be labeled after the prefix defined above.

After labeling the vertices at distance most h' from v in C, we must have labelled the vertex $c_{h'}'$, which is the vertex such that $(L_{h'+1}(v) \setminus C) \cap N(c_{h'}')$ ($c_{h'}'$ exists by Lemma 6.2.5). Therefore we can choose the vertices of $L_{h'+1}(v) \setminus C$ next in some order. This concludes the induction.

Finally, we can visit the remaining vertices of $N(C)$: each vertex of $N(C)$ has a maximal label from a neighbour in C which v does not have, so we need not choose v before any vertex of S. □

Lemma 6.6.4. *Let G be a connected bipartite permutation graph with two non-trivial components C_1 and C_2 of $G \setminus N[v]$ for some $v \in V(G)$. Suppose that there is a vertex $y \in N(v)$ that is in every substar of v and that $N[C_1]$ has been numbered by MNS along with all trivial components of $G \setminus N[v]$. Finally, let $S' \subseteq N(C_2)$ be the unnumbered vertices of $N(C_2)$.*

If y was numbered first by MNS among all vertices of $N(C_2)$, v is not numbered, and no vertex in C_2 is numbered, then $C_2 \cup S'$ can be the next $|C_2 \cup S'|$ vertices numbered by MNS.

Proof. Let q be a vertex in C_2 as far away from v as possible, and let $h' = d_G(v, q)$; note that $h' \geq 3$ as otherwise C_2 is not a non-trivial component. For $2 \leq h \leq h'$, let $c_h \in L_h(v) \setminus C_2$ be such that $(L_{h-1}(v) \setminus N[C_2]) \cap N(c_h)$; that is, c_h is adjacent to all vertices of $N[C_2]$ on the layer above c_h in $N[C_2]$. Such a vertex always exists by Lemma 6.2.4.

MNS must choose a next vertex whose label is maximal; it must therefore have a neighbour in $N(C_1) \cup N(C_2) \cup T$ where T is the set of trivial components of $G \setminus N[v]$. By

definition, $N(C_2)$ $N(c_2)$, and therefore c_2 can be chosen next by MNS as no vertex in $L_2(v) \cap (T \cup N(C_1)) \cup \{v\}$ has a label yet.

Now we proceed as in the proof of Lemma 6.6.3. The proof is by induction on h'. In the base case, suppose that $h' = 3$. After choosing c_2, each vertex in S' receives the number given to c_2, as does c_3. Since we have numbered the vertices of T, we may not be able to choose c_3 before some vertices of S'; label as many vertices of S' as required by MNS before we can choose c_3. Since $c_3 \in L_3(v)$ and S' $L_1(v)$, the vertices on $L_3(v) \setminus C_2$ never have their label updated by visiting vertices of S'. Therefore, we can choose c_3 next. Then, we can label the vertices of $L_2(v) \setminus C_2$ as required until we can label the vertex c_2', where c_2' is such that $(L_3(v) \setminus C_2)$ $N(c_2')$ if it was not already labeled. We can then finish labelling all vertices of $L_2(v) \setminus C_2$ in some order: all such vertices have maximal labels with the number given to c_3. Then, all vertices of $L_3(v) \setminus C_2$ can be numbered in some order, as all of their neighbours have been visited.

Therefore, assume that the result holds for h' 3 and consider $h'+1$. By induction, all vertices of C at distance at most h' from v can be labeled after the prefix defined above.

After labeling the vertices at distance most h' from v in C, we must have labelled the vertex $c'_{h'}$ such that $(L_{h'+1}(v) \setminus C_2)$ $N(c'_{h'})$: therefore we can choose the vertices of $L_{h'+1}(v) \setminus C_2$ next in some order. This concludes the induction.

Finally, we can visit vertices of S' which have not yet been labelled: each vertex of S' has a maximal label from a neighbour in C_2 which v does not have, so we need not choose v before any vertex of S'. □

Lemma 6.6.5. *Let G be a connected bipartite permutation graph, and let v be a vertex of G. If v is not a cut vertex and there is a vertex $y \in N(v)$ that is in every substar of v, then v is an MNS end-vertex.*

Proof. Start an MNS ordering at y. Let T be the trivial components of G $N[v]$; after visiting y, visit all vertices of T in any order.

By Lemma 6.2.7, there are at most two non-trivial components of G $N[v]$, say C_1 and C_2. By Lemma 6.2.4, there is a vertex $c_1 \in C_1$ such that $N(C_1)$ $N(c_1)$ and there is a vertex $c_2 \in C_2$ such that $N(C_2)$ $N(c_2)$.

By assumption that y is in every substar of v, $(y;v);(y;c_1);(y;c_2) \in E(G)$, and after choosing y, $\grave{}(c_1) = \grave{}(c_2) = \grave{}(v)$. By Lemma 6.6.3, we can visit all of C_1 next along with $N(C_1)$. By Lemma 6.6.4, we can visit all of C_2 next along with $N(C_2)$.

Since v is not a cut vertex, there are no vertices of $N(v)$ which are left unnumbered. Finally, we can finish the ordering with v, as required. □

(a) A triple of vertices in an ordering with $a < b < c$.

(b) A required neighbor $d < b$; it is not known whether $d < a$ or $a < d$.

Figure 6.12: The situation to consider for characterizing orderings generated by MNS. A solid line indicates that the edge is present, while a dashed line indicates the absence of a edge. There may be other vertices between those shown, i.e., $\{a; b; c\}$ need not be consecutive. Vertices with no lines between them may or may not be adjacent.

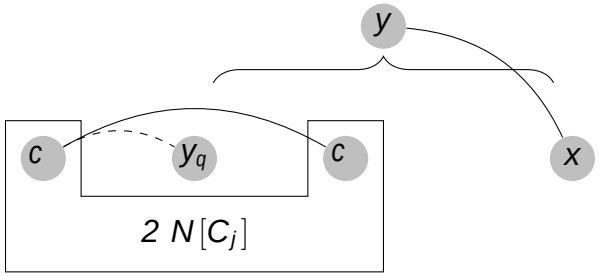
The remainder of the section is dedicated to showing that the condition is necessary. We will make use of the following theorem for MNS, which is illustrated in Figure 6.12. In fact, all search algorithms have a similar theorem which characterizes the orderings they generate, but we will only use this one. Such a characterization is called a *vertex ordering characterization*; see e.g., Corneil and Stacho [38] for more on this topic.

Theorem 6.6.6 (Theorem 2.8, Corneil and Krueger [34])**.** *Given an MNS ordering α of $G = (V; E)$, if $a < b < c$ and $(a; c) \in E$ and $(a; b) \notin E$, then there exists a vertex d with $d < b$ such that $(d; b) \in E$ and $(d; c) \notin E$.*

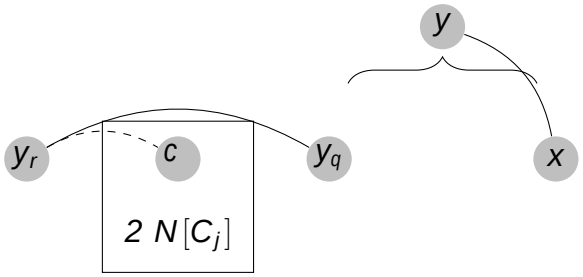
We first establish the following consequences of Theorem 6.6.6. Both of the following lemmas apply Theorem 6.6.6 in some specific conditions. They are intended to be used alternately. For example, if Lemma 6.6.7 applies, then afterwards, Lemma 6.6.8 will apply, and after that, Lemma 6.6.7 will once again apply. Thus, if you can prove that one of these specific cases is met, one can alternate these lemmas indefinitely. This enables a contradiction in the proof of Lemma 6.6.9, the necessity condition for v to be an MNS end-vertex. Specifically, the contradiction will be to the fact that the graph we are considering is finite. The conditions for these lemmas are illustrated in Figure 6.13.

Lemma 6.6.7. *Let G be a graph and let $v \in V(G)$. Suppose that α is an MNS ordering of G ending at v, and that $y \in N(v)$ is leftmost in α. Let $C_1, \ldots, C_t$ be the components of $G - N[v]$. Suppose that C_j is such that $y \notin N(C_j)$.*

If $c < y_q < c'$ for some $c \in C_j$, $c' \in N[C_j]$, and $y_q \notin N[C_j]$, where $(c, c') \in E(G)$,



(a) The situation required to apply Lemma 6.6.7.



(b) The situation required to apply Lemma 6.6.8.

Figure 6.13: The situation to consider for Lemmas 6.6.7 and 6.6.8.

$(y_q; c\ ) \not\in E(G)$, and $y_q \quad y$, then there is a vertex $y_r < c$ such that $y_r \not\in N[C_j]$. Moreover, there is a vertex y_q' with $c < y_q' \quad y_q$ such that $(y_r; y_q') \in E(G)$.

Proof. We can apply Theorem 6.6.6 to $(c\ ; y_q; c\ )$ to get a vertex y_r^0 such that $(y_r^0; y_q) \in E(G)$ but $(y_r^0; c\ ) \not\in E(G)$ and $y_r^0 < y_q$. Since $y_q < v$ (as v is last), $y_r^0 \neq v$. Therefore $y_r^0 < y$, $y_r^0 \not\in N(v)$, and we have that $y_r^0 \not\in N[C_j]$.

If $y_r^0 < c < y_q < c$, then we are done by taking $y_r = y_r^0$ and $y_q' = y_q$.

Since $y_r^0 \not\in N[C_j]$, we have that $(c\ ; y_r^0) \not\in E(G)$. Moreover, it is still the case that $(c\ ; c\ ) \in E(G)$. Starting with $i = 0$, we can apply Theorem 6.6.6 to $(c\ ; y_r^i; c\ )$ to get a vertex y_r^{i+1} such that $(y_r^{i+1}; y_r^i) \in E(G)$ but $(y_r^{i+1}; c\ ) \not\in E(G)$ and $y_r^{i+1} < y_r^i$.

Since $y_r^{i+1} < y$, $y_r^{i+1} \not\in N(v)$, and therefore $y_r^{i+1} \not\in N[C_j]$.

While y_r^{i+1} is such that $c < y_r^{i+1} < y_r^i < c$, we can continue to apply Theorem 6.6.6 to $(c\ ; y_r^i; c\ )$ after updating i to have value $i + 1$. Since G is finite, we cannot apply Theorem 6.6.6 indefinitely, and therefore we must eventually get a vertex y_r^{i+1} such that $y_r^{i+1} < c$. Take $y_r = y_r^{i+1}$ and $y_q' = y_r^i$ to complete the proof. □

Lemma 6.6.8. *Let G be a graph and let $v \in V(G)$. Suppose that is an MNS ordering of G ending at v, and that $y \in N(v)$ is leftmost in . Let $C_1, \ldots, C_t$ be the components of $G \quad N[v]$. Suppose that C_j is such that $y \not\in N(C_j)$.*

If $y_r < c < y_q$ for some $c \in C_j$, and $y_r; y_q \not\in N[C_j]$, where $(y_r; y_q) \in E(G)$ and $(y_r; c\ ) \not\in E(G)$, then there is a vertex $c < y_r$ such that $c \in N[C_j]$. Moreover, there is a vertex c' with $y_r < c' \quad c$ such that $(c\ ; c'\ ) \in E(G)$.

Proof. We can apply Theorem 6.6.6 to $(y_r; c\ ; y_q)$ to get a vertex c^0 such that $(c^0; c\ ) \in E(G)$ but $(c^0; y_q) \not\in E(G)$ and $c^0 < c$. Since $y_q < v$ (as v is last), $c^0 \neq v$. Since $(c^0; c\ ) \in E(G)$, $c^0 \in N[C_j]$.

If $c^0 < y_r < c < y_q$, then we are done by taking $c = c^0$ and $c' = c$

Since $c^0 \in N[C_j]$, we have that $(c^0; y_r) \not\in E(G)$. Moreover, it is still the case that $(y_r; y_q) \in E(G)$. Starting with $i = 0$, we can apply Theorem 6.6.6 to $(y_r; c^0; y_q)$ to get a vertex c^{i+1} such that $(c^{i+1}; c^i\ ) \in E(G)$ but $(c^{i+1}; y_q) \not\in E(G)$ and $c^{i+1} < c^i$.

Since $c^{i+1} \in N[C_j]$, $(c^{i+1}; y_r) \not\in E(G)$.

While c^{i+1} is such that $y_r < c^{i+1} < c^i < c$, we can continue to apply Theorem 6.6.6 to $(y_r; c^i\ ; y_q)$ after updating i to have value $i + 1$. Since G is finite, we cannot apply Theorem 6.6.6 indefinitely, and therefore we must eventually get a vertex c^{i+1} such that $c^{i+1} < y_r$. Take $c = c^{i+1}$ and $c' = c^i$ to complete the proof. □

Lemma 6.6.9. *Let G be a connected bipartite graph and let v be a vertex of G. If v is an MNS end-vertex, then v is not a cut vertex and there is a vertex $y \in N(v)$ that is in every substar of v.*

Proof. Let be an MNS ordering of G ending at v.

Since every MNS search ordering is a generic search ordering, v is not a cut vertex by Theorem 6.6.2.

Let $C_1, \ldots, C_t$ be the connected components of G $N[v]$. If t 1, then the lemma holds trivially. If $t = 2$, then since v is not a cut vertex and there are no edges between vertices in $N(v)$ by Observation 6.2.3, the lemma must hold. We therefore assume that t 3.

Let $y \in N(v)$ be leftmost in . We claim that y must be adjacent to all components of G $N[v]$. Suppose the contrary that this is not the case, and let C_j be such that $y \notin N_G(C_j)$.

Let $c \in N_G(C_j)$ $N(v)$ be leftmost in . Since y is leftmost among all vertices of $N(v)$ in , we have that $y <$ c.

Claim 6.6.10. *There is no vertex $c_1 \in C_j$ such that $c_1 <$ c and $d(c_1, c) = 1$.*

Proof of claim: Suppose to the contrary that there is a vertex $c_1 \in C_j$ such that $c_1 <$ c and $d(c_1, c) = 1$. Let $c_0 = c$.

There are two cases, based on where c_1 is relative to y.

Case 1: $c_1 < y <$ c_0. Since $(c_1, c_0) \in E(G)$ but $(y, c_1) \notin E(G)$ (because y is not adjacent to C_j), we can apply Lemma 6.6.7 to (c_1, y, c_0). Then, we can alternate applications of Lemmas 6.6.8 and 6.6.7 to establish that must be infinitely long, a contradiction to the fact that G is finite.

Case 2: $y <$ $c_1 <$ c_0. Since v is the last vertex of , we have $y <$ $c_1 <$ $c_0 <$ v. Since $y \in N(v)$, $(y, v) \in E(G)$; since $c_1 \in C_j$ and y is not adjacent to C_j, $(y, c_1) \notin E(G)$. Therefore we can apply Lemma 6.6.8 to (y, c_1, c_0). Then, we can alternate applications of Lemmas 6.6.7 and 6.6.8 to establish that must be infinitely long, a contradiction to the fact that G is finite.

Since we have a contradiction in either case, the lemma is proved.

Claim 6.6.11. *If there is no vertex $c_i \in C_j$ such that $c_i <$ c and $d(c_i, c) = i$, then there is no vertex $c_{i+1} \in C_j$ such that $c_{i+1} <$ c and $d(c_{i+1}, c) = i + 1$.*

Proof of claim: Suppose to the contrary that there is a vertex $c_{i+1} \in C_j$ such that $c_{i+1} < c$ and $d(c_{i+1}; c) = i + 1$ but for all $c_i \in C_j$ such that $d(c_i; c) = i$, $c < c_i$. Let $c_0 = c$.

There are two cases, based on where c_{i+1} is relative to y.

Case 1: $c_{i+1} < y < c_0 < c_i$. Since $(c_{i+1}; c_i) \in E(G)$ but $(y; c_{i+1}) \notin E(G)$ (because y is not adjacent to C_j), we can apply Lemma 6.6.7. Then, we can alternate applications of Lemmas 6.6.8 and 6.6.7 to establish that must be infinitely long, a contradiction to the fact that G is finite.

Case 2: $y < c_{i+1} < c_0 < c_i$. Since v is the last vertex of , we have $y < c_{i+1} < c_0 < c_i < v$. Since $y \in N(v)$, $(y; v) \in E(G)$; since $c_{i+1} \in C_j$ and y is not adjacent to C_j, $(y; c_{i+1}) \notin E(G)$. Therefore we can apply Lemma 6.6.8. Then, we can alternate applications of Lemmas 6.6.7 and 6.6.8 to establish that must be infinitely long, a contradiction to the fact that G is finite.

Since we have a contradiction in either case, the lemma is proved.

Therefore, for all $c' \in C_j$, $c < c'$. Since v is the end vertex, $y < c < c' < v$. Let $c' \in C_i$ be leftmost in . When c' is chosen by MNS, $l(c) = S$ for some non-empty set S. Since c' is leftmost, it must have received a number in its label from a neighbour of v. Let $T = \{t \in N(v) | (t) \in S\}$ be the neighbours of v contributing to the label of c'. Since S is non-empty, T is non-empty as well. However, since $y < c$, $(y; v) \in E(G)$, and $(t; v) \in E(G)$ for all $t \in T$, $(y) \in l(v)$ and $(t) \in l(v)$ for all $t \in T$ when c' is chosen. Therefore, $l(c') \subset l(v)$ at this step, and MNS must choose v over c'. That is, $y < v < c'$, contradicting that v was the last vertex of .

□

It is now clear that Theorem 6.6.1 follows from Lemmas 6.6.5 and 6.6.9; therefore we also have the following corollary.

Corollary 6.6.12. *The* MNS-End-Vertex *problem for bipartite permutation graphs is in* P.

Proof. Algorithm 6.6.1 solves MNS-End-Vertex. In polynomial time, we can compute the substars of $G \setminus N[v]$ for a candidate end-vertex v. Also in polynomial time, we can determine if there is a vertex of $N(v)$ which is present in every substar of $G \setminus N[v]$. By Theorem 6.6.1, we can use this information to determine if v is an MNS end-vertex. □

Algorithm 6.6.1: Polynomial Time MNS-End-Vertex solution for Bipartite Permutation Graphs.

Input: A bipartite permutation graph $G = (V; E)$ and a vertex $v \; 2 \; V$.

Output: `true` if there is a MNS ordering of G ending at v, or `false` otherwise.

```
1 Compute the components C_1; : : : ; C_k of G    N[v];
2 for w 2 N(v) do
3 |   s    true;
4 |   for i = 1; i < k + 1; i + + do
5 |   |   if w 2 N(C_i) then s    false;
6 |   if s then return true;
7 return false;
```

Chapter 7

Conclusions and Future Work

We conclude the thesis with a summary of main results and a discussion on future work.

7.1 Summary of Main Results

This thesis has aimed to relate various orderings of graphs. Focusing on two types of problems, the thesis attempts to answer two main questions, each related to a type of ordering problem. First, do "linear" graphs have efficient algorithms for solving ordering problems? Second, can "linear" graphs be traversed efficiently so that a particular vertex is last?

In this thesis, we considered AT-free graphs to be "linear". This is due to their generalization of interval graphs (see, e.g., Corneil and Stacho [38]), which can be seen as generalizations of paths, which are very linear. This thesis dealt with superfragile graphs and proper interval k-trees, and bipartite permutation graphs which are all subsets of AT-free graphs.

The first question attempts to determine if this linear structure assists with solving Type I problems. From the existing work on Optimal Linear Arrangement, we know that the answer to this question is "not always" (unless P = NP). In particular, we know that Optimal Linear Arrangement is NP-complete on interval graphs (Cohen et al. [29]). However, when efficient algorithms are possible for one problem on such a graph, other problems can also be tackled using similar ideas. Thus, the intuition that "linear graphs" yield efficient algorithms for linear ordering problems should be modified.

Rather than expecting efficient algorithms on these classes of graphs for Type I problems, one should expect similarities between problems on "linear graphs".

We explored this modified intuition on AT-free graphs by connecting the Type I problems considered. First, we connected optimal orderings for Cutwidth to those for Imbalance (Lemma 3.4.2). We showed that when an optimal ordering for Cutwidth has a nice property regarding the ranks of the vertices, this ordering is immediately optimal for Imbalance as well. Exploring this connection further, the remainder of Section 3.4 showed thinking about *both* Cutwidth and Imbalance was helpful to prove results about optimal orderings for both of these problems. This connection also yielded closed formulas for the graph classes in Section 4.3. When we considered the additional structure imposed on graphs by being AT-free, we were able to show that proper interval k-trees have imbalance-minimal orderings which are also cutwidth-minimal and optimal for Optimal Linear Arrangement all at once (Corollary 4.6.12). In the case of superfragile graphs, there is an ordering which is optimal for both Imbalance and Optimal Linear Arrangement at the same time (Theorems 4.5.2 and 5.3.1). Section 4.8 discussed how others may be able to accomplish this goal for additional classes of AT-free graphs. These results encourage the consideration of all Type I problems simultaneously for a given class of graphs when efficient algorithms are possible for any single problem.

The study of these problems on AT-free graphs led to general results. We were able to prove that for *any* graph, there is an imbalance-minimal or cutwidth-minimal ordering where each equivalence class of true twins appears consecutively (Theorems 3.5.1 and 4.4.2). This result may be very helpful on other subclasses of AT-free graphs (e.g., for threshold graphs), but also allowed progress in the fixed-parameter setting. These theorems show that for these problems, vertices that are true twins "behave" similarly, enabling us to treat equivalence classes of true twins as single entities. Since each class may be arbitrarily large, this allows us to reduce the complexity of problems in cases where there are many (or large) classes of true twins.

We are able to solve these Type I problems efficiently for more graph parameters by observing how true twins behave. First, we showed that Cutwidth has FPT algorithms for the parameters edge clique cover number (Theorem 3.6.1). Then, by defining the restricted twin cover number of a graph, we generalized the of the vertex cover number to tackle denser graph classes. Using this new parameter, we were able to solve both Cutwidth and Optimal Linear Arrangement with a parameter more general than the vertex cover number (Theorems 3.7.5 and 5.4.5). We can now efficiently solve Optimal Linear Arrangement with a parameter than allows denser graphs than vertex cover number does; this is a feat that had not been done before. Thus, we are building towards an understanding of these problems with parameters which are suitable for even more dense

graphs (like clique width). This understanding comes from a combination of the behaviour of true twins and twin covers on these graphs, which often apply techniques introduced for vertex cover number parameterizations.

Studying vertex cover number parameterizations continues to be helpful. In addition to adapting the work of Fellows et al. [52] and Lokshsatnov [103] to graphs with bounded restricted twin cover number, we expanded the results of Cygan et al. [42]. We applied their ideas for Cutwidth to Imbalance for two results. First, we obtained an FPT algorithm for Imbalance with the vertex cover number parameter (Theorem 4.7.9). This algorithm improves the result of Fellows et al. [52] for Imbalance. Second, we were able to show that under reasonable complexity assumptions, Imbalance does not have a polynomially sized kernel when the parameter is the vertex cover number of the graph (Theorem 4.7.5). This work provides more evidence that there are connections between Type I problems, including connections which may arise from studying how vertex covers affect the solutions to these problems.

Turning to the second main question, we are interested in if linear structure enables placing a vertex last by a search algorithm. For bipartite permutation graphs, a subset of AT-free graphs that we studied for this problem in the thesis, the answer appears to be "yes". Here, our intuition regarding the "linear" structure of the graph was correct for Type II problems. By understanding the AT-free structure of the graph, we were able to characterize vertices which appear last for most search algorithms. Bipartite permutation graphs were shown to have at most two deep components after removing the closed neighbourhood of a vertex. The components can be thought of as the "ends" of the graph relative to the vertex removed to obtain them (if there is only one such component, the vertex removed is "near" one end).

Understanding these components and how they are connected was immensely useful. Depending on the search algorithm, we needed to determine where the target end-vertex would be: in such a component, or the vertex to be removed to create them. In the case of layer searches, like BFS or LBFS, a candidate end-vertex must have been in a deep component. Characterizing the vertices to remove in order to obtain such a deep component yielded efficient algorithms for BFS-End-Vertex (polynomial, Theorem 6.3.9) and LBFS-End-Vertex (linear, Theorem 6.4.1). For other searches, like DFS and MNS, we showed that one would need to "skip" the candidate vertex until the rest of the graph was searched. In particular, by characterizing the end-vertices for these searches, we know what the neighbourhoods of an end-vertex need to have for both of these searches. In turn, we found efficient algorithms for both DFS-End-Vertex (linear, Theorem 6.5.7) and MNS-End-Vertex (polynomial, Corollary 6.6.12). From the background work discussed in Section 6.1, we know that interval graphs yield to efficient algorithms. This

thesis showed that, for all but MCS and LDFS, the bipartite analog of interval graphs have efficient algorithms too.

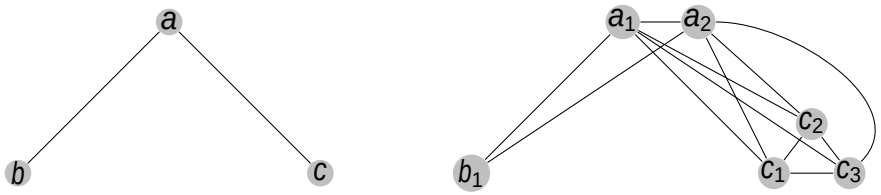
When solving ordering related problems on AT-free graphs, one should look for similarities with other problems. In the case of Type I problems, connections to other problems have been shown and should henceforth be sought out. These connections have been shown to be useful for proving new results and exploring optimal orderings. Such connections may arise from the behaviour of true twins, vertex covers, or a lack of asteroidal triples. For Type II problems, understanding the linear structure of the graphs has been shown to be fruitful. The linear structure can be understood by looking at the layers of the graph, and understanding where candidate vertices are in those layers.

7.2 Future Work

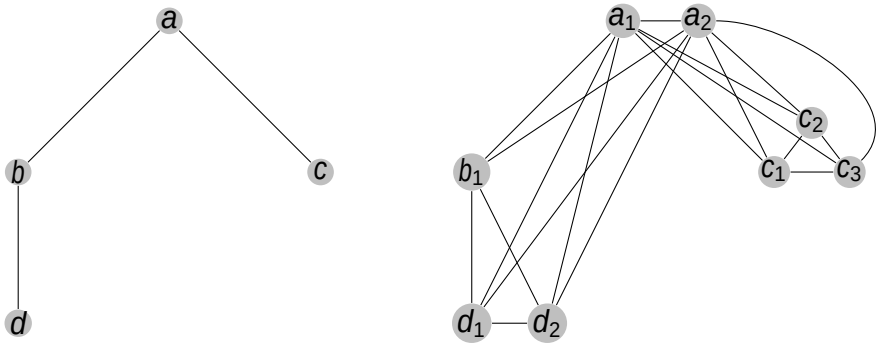
Type I Problems

There are several concrete problems which naturally lend themselves to future work. First, there are some straightforward questions regarding some classes of graphs:

- What is the complexity of Cutwidth, Imbalance, and Optimal Linear Arrangement on the classes for which these problems have no known complexity? (Figures 3.1a, 4.1a, and 5.1a.)
- What is the complexity of Cutwidth, Imbalance, and Optimal Linear Arrangement on *trivially perfect graphs*? Trivially perfect graphs are interval graphs for which there is a representation of the graph where, for each pair of intervals, one contains the other or the two are disjoint. These graphs are a generalization of superfragile graphs. One can obtain a connected superfragile graph by taking a star and replacing each vertex of the star with a clique, and adding all edges from one clique to another if the corresponding vertices for the cliques were adjacent in the star. Trivially perfect graphs are similar, except that one uses a rooted tree, rather than a star, and all edges are added to any clique on a path to the root (see Figure 7.1). Trivially perfect graphs are a natural next step after establishing algorithms for superfragile graphs. Preliminary exhaustive computer searches show that there is always an ordering which is optimal for all three problems simultaneously when these graphs are small (when the graphs are large, exhaustive search takes too long to verify this claim).



(a) A superfragile graph (right) created by replacing vertices of a (rooted) star (left) with a clique.



(b) A trivially perfect graph (right) created by replacing vertices of a rooted tree (left) with a clique. After replacing vertices with cliques, all edges are also added to any clique on the shortest path to the root.

Figure 7.1: The similarities between superfragile graphs and trivially perfect graphs.

- The positive superfragile results encourage the investigation of these problems in the "reverse" direction. Knowing that the answer to the first question investigated by this thesis may be "not always", one wonders if we can say more. Since Optimal Linear Arrangement is NP-complete on interval graphs, are Cutwidth and Imbalance also NP-complete on interval graphs? The previous discussion notes that efficient algorithms on Cutwidth inspired efficient algorithms on Imbalance and Optimal Linear Arrangement. Do hardness results for Optimal Linear Arrangement inspire hardness results for Imbalance and Cutwidth?
- Is an $(n;k)$ proper interval graph an edge-maximal graph with n vertices of bandwidth at most k?

Second, there are some unresolved questions in the FPT setting:

- Does Imbalance have a polynomial-sized kernel when the parameter is the solution

size?

- Can the algorithm of Theorem 4.7.9 and its corresponding algorithm for Cutwidth (Cygan et al. [42]) be used to speed up parameterizations of these problems for graphs of bounded restricted twin cover number? As this does not yield a new FPT result for Imbalance (it might only be faster), this question is targeted towards Cutwidth.
- Does Optimal Linear Arrangement admit an FPT algorithm when the parameter is the treewidth of the graph (and maybe the maximum degree)?
- Another generalization of the twin cover number and neighbourhood diversity number of a class of graph is the *shrub depth* (denoted $sd(G)$) (Ganian et al. [59]. The definition is rather technical (and we refer the reader to Ganian et al. [59] for it), but the relation to other parameters studied in this work is shown in Figure 7.2. What is the complexity of Imbalance on classes of graphs with bounded shrub depth?

More generally, are these approaches helpful for other graph ordering problems? For example, Fellows et al. [52] also use an ILP for a vertex-cover parameterization for Bandwidth and Distortion (see, e.g., Fellows et al. [52] for a definition). However, since there are results for both of these problems on bipartite permutation graphs and threshold graphs (Heggernes et al. [76], Heggernes et al. [78], Uehara [135]), progress may only be possible in the parameterized setting. Díaz [44] lists other ordering problems with numerical optimization functions (i.e., Type I problems) that may be less studied.

Perhaps the most challenging direction for future work would be to determine if there exists a common framework for describing Cutwidth, Imbalance, and Optimal Linear Arrangement which could be used to prove complexity results about all of them simultaneously. If such a framework is not feasible, perhaps more results like Lemma 3.4.2 may still be of help. An explicit connection that that relates Imbalance and Optimal Linear Arrangement is missing in general.

Type II Problems

The most obvious directions for future work for the S-End-Vertex are very concrete, and are based on filling in Table 6.1.

- Is S-End-Vertex in P for LDFS and MCS on bipartite permutation graphs?

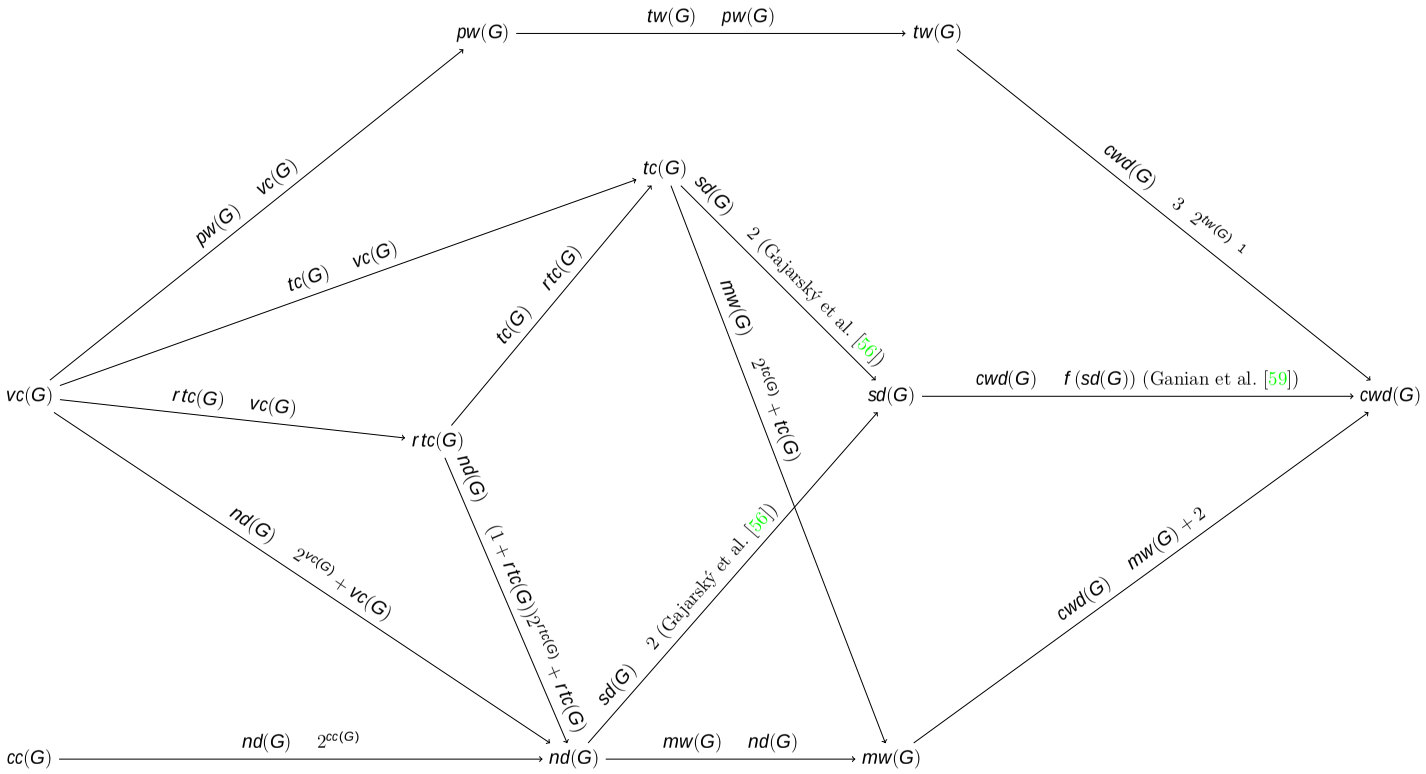


Figure 7.2: The relationship between shrub depth (Ganian et al. [59]) and other parameters for graph problems.

- Is MNS-End-Vertex NP-complete for general bipartite graphs?

Table 6.1 does not mention general AT-free graphs. As suggested in the previous section, considering these general classes of "linear" graphs will provide a much more complete understanding of S-End-Vertex.

There are also some directions for future work that are unrelated to Table 6.1.

- Are there any FPT results for these problems? The work of Kratsch et al. [92] showed that BFS-End-Vertex has a $O(2^n)$ algorithm on general graphs—can this be improved to polynomial in n but exponential in some parameter k?
- The proofs in e.g., Section 6.3, only rely on the fact that Lemma 6.2.4 hold, the graphs are bipartite, and the graphs have at most two deep components after removing a vertex. Are these properties enough to characterize bipartite permutation graphs? Are there other classes of graphs where related properties hold and we can obtain similar end-vertex characterizations?
- Do Algorithms 6.3.1, 6.4.1, 6.5.1, or 6.6.1 have any other applications? For example, can one of them be used to test if a graph is a bipartite permutation graph?

References

[1] D. Adolphson and T.C. Hu. Optimal linear ordering. *SIAM Journal on Applied Mathematics*, 25(3):403–423, 1973.

[2] Donald L. Adolphson. Single machine job sequencing with precedence constraints. *SIAM Journal on Computing*, 6(1):40–54, 1977.

[3] Sanjeev Arora, Alan M. Frieze, and Haim Kaplan. A new rounding procedure for the assignment problem with applications to dense graph arrangement problems. *Mathematical Programming*, 92(1):1–36, 2002.

[4] Olav Røthe Bakken. Arrangement problems parameterized by neighbourhood diversity. Master's thesis, University of Bergen, 2018.

[5] Jesse Beisegel, Carolin Denkert, Ekkehard Köhler, Matjaz Krnc, Nevena Pivac, Robert Scheffler, and Martin Strehler. On the end-vertex problem of graph searches. *Discrete Mathematics & Theoretical Computer Science*, 21(1), 2019.

[6] Anne Berry, Jean R.S. Blair, Jean-Paul Bordat, and Genevieve Simonet. Graph extremities defined by search algorithms. *Algorithms*, 3(2):100–124, 2010.

[7] Anne Berry and Jean Paul Bordat. Local LexBFS properties in an arbitrary graph. *Proceedings of Journees Informatiques Messines*, 2000.

[8] Therese Biedl, Timothy Chan, Yashar Ganjali, Mohammad Taghi Hajiaghayi, and David R. Wood. Balanced vertex-orderings of graphs. *Discrete Applied Mathematics*, 148(1):27–48, 2005.

[9] Guillaume Blin, Guillaume Fertin, Danny Hermelin, and Stéphane Vialette. Fixed-parameter algorithms for protein similarity search under mRNA structure constraints. *Journal of Discrete Algorithms*, 6(4):618–626, 2008.

[10] Hans L. Bodlaender. A partial *k*-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1-2):1–45, 1998.

[11] Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *Journal of Computer and System Sciences*, 75(8):423–434, 2009.

[12] Hans L. Bodlaender, Michael R. Fellows, and Dimitrios M. Thilikos. Starting with nondeterminism: The systematic derivation of linear-time graph layout algorithms. In Branislav Rovan and Peter Vojtás, editors, *Mathematical Foundations of Computer Science 2003, 28th International Symposium, MFCS 2003, Bratislava, Slovakia, August 25-29, 2003, Proceedings*, volume 2747 of *Lecture Notes in Computer Science*, pages 239–248. Springer, 2003.

[13] Hans L. Bodlaender, Stéphan Thomassé, and Anders Yeo. Kernel bounds for disjoint cycles and disjoint paths. *Theoretical Computer Science*, 412(35):4570–4578, 2011.

[14] J.A. Bondy and U.S.R. Murty. *Graph Theory*. Springer, 2008.

[15] Rodrigo A. Botafogo. Cluster analysis for hypertext systems. In Robert Korfhage, Edie M. Rasmussen, and Peter Willett, editors, *Proceedings of the 16th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval. Pittsburgh, PA, USA, June 27 - July 1, 1993*, pages 116–125. ACM, 1993.

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

[17] Andreas Brandstädt and Dieter Kratsch. On the restriction of some NP-complete graph problems to permutation graphs. In Lothar Budach, editor, *Fundamentals of Computation Theory, FCT '85, Cottbus, GDR, September 9-13, 1985*, volume 199 of *Lecture Notes in Computer Science*, pages 53–62. Springer, 1985.

[18] Andreas Brandstädt, Jeremy P. Spinard, and Van Bang Le. *Graph Classes: A Survey*. SIAM, 1999.

[19] Melvin A. Breuer. A class of min-cut placement algorithms. In Judith G. Brinsfield, Stephen A. Szygenda, and David W. Hightower, editors, *Proceedings of the 14th Design Automation Conference, DAC '77, New Orleans, Louisiana, USA, June 20-22, 1977*, pages 284–290. ACM, 1977.

[20] Jin-yi Cai, Venkatesan T. Chakaravarthy, Lane A. Hemaspaandra, and Mitsunori Ogihara. Competing provers yield improved Karp-Lipton collapse results. *Information and Computation*, 198(1):1–23, 2005.

[21] Yixin Cao, Zhifeng Wang, Guozhen Rong, and Jianxin Wang. Graph searches and their end vertices. In Pinyan Lu and Guochuan Zhang, editors, *30th International Symposium on Algorithms and Computation, ISAAC 2019, December 8-11, 2019, Shanghai University of Finance and Economics, Shanghai, China*, volume 149 of *LIPIcs*, pages 1:1–1:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[22] Jou-Ming Chang, Chin-Wen Ho, and Ming-Tat Ko. LexBFS-ordering in asteroidal triple-free graphs. In Alok Aggarwal and C. Pandu Rangan, editors, *Algorithms and Computation, 10th International Symposium, ISAAC '99, Chennai, India, December 16-18, 1999, Proceedings*, volume 1741 of *Lecture Notes in Computer Science*, pages 163–172. Springer, 1999.

[23] Pierre Charbit, Michel Habib, and Antoine Mamcarz. Influence of the tie-break rule on the end-vertex problem. *Discrete Mathematics & Theoretical Computer Science*, 16(2):57–72, 2014.

[24] Pierre Charbit, Michel Habib, Lalla Mouatadid, and Reza Naserasr. A new graph parameter to measure linearity. In Xiaofeng Gao, Hongwei Du, and Meng Han, editors, *Combinatorial Optimization and Applications - 11th International Conference, COCOA 2017, Shanghai, China, December 16-18, 2017, Proceedings, Part II*, volume 10628 of *Lecture Notes in Computer Science*, pages 154–168. Springer, 2017.

[25] Fan R.K. Chung. On the cutwidth and the topological bandwidth of a tree. *SIAM Journal on Algebraic Discrete Methods*, 6(2):268–277, 1985.

[26] Fan R.K. Chung. Labelings of graphs. In Lowell W. Beineke and Robin J. Wilson, editors, *Selected Topics in Graph Theory*, volume 3, chapter 7, pages 151–168. Academic Press, San Diego, CA, 1988.

[27] Fan R.K. Chung and Paul D. Seymour. Graphs with small bandwidth and cutwidth. *Discrete Mathematics*, 75(1-3):113–119, 1989.

[28] Moon-Jung Chung, Fillia Makedon, Ivan Hal Sudborough, and Jonathan S. Turner. Polynomial time algorithms for the min cut problem on degree restricted trees. *SIAM Journal on Computing*, 14(1):158–177, 1985.

[29] Johanne Cohen, Fedor V. Fomin, Pinar Heggernes, Dieter Kratsch, and Gregory Kucherov. Optimal linear arrangement of interval graphs. In Rastislav Kralovic and Pawel Urzyczyn, editors, *Mathematical Foundations of Computer Science 2006, 31st International Symposium, MFCS 2006, Stara Lesna, Slovakia, August 28-September 1, 2006, Proceedings*, volume 4162 of *Lecture Notes in Computer Science*, pages 267–279. Springer, 2006.

[30] Derek G. Corneil. Lexicographic breadth first search - A survey. In Juraj Hromkovic, Manfred Nagl, and Bernhard Westfechtel, editors, *Graph-Theoretic Concepts in Computer Science, 30th International Workshop,WG 2004, Bad Honnef, Germany, June 21-23, 2004, Revised Papers*, volume 3353 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2004.

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

[32] Derek G. Corneil, Feodor F. Dragan, Michel Habib, and Christophe Paul. Diameter determination on restricted graph families. *Discrete Applied Mathematics*, 113(2-3):143–166, 2001.

[33] Derek G. Corneil, Ekkehard Köhler, and Jean-Marc Lanlignel. On end-vertices of lexicographic breadth first searches. *Discrete Applied Mathematics*, 158(5):434–443, 2010.

[34] Derek G. Corneil and Richard M. Krueger. A unified view of graph searching. *SIAM Journal on Discrete Mathematics*, 22(4):1259–1276, 2008.

[35] Derek G. Corneil, Stephan Olariu, and Lorna Stewart. Linear time algorithms for dominating pairs in asteroidal triple-free graphs. *SIAM Journal on Computing*, 28(4):1284–1297, 1999.

[36] Derek G. Corneil, Stephan Olariu, and Lorna Stewart. The LBFS structure and recognition of interval graphs. *SIAM Journal on Discrete Mathematics*, 23(4):1905–1953, 2009.

[37] Derek G. Corneil and Udi Rotics. On the relationship between clique-width and treewidth. *SIAM Journal on Computing*, 34(4):825–847, 2005.

[38] Derek G. Corneil and Juraj Stacho. Vertex ordering characterizations of graphs of bounded asteroidal number. *Journal of Graph Theory*, 78(1):61–79, 2015.

[39] Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Computing Systems*, 33(2):125–150, 2000.

[40] Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77–114, 2000.

[41] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

[42] Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. On cutwidth parameterized by vertex cover. *Algorithmica*, 68(4):940–953, 2014.

[43] Josep Díaz, Mathew D. Penrose, Jordi Petit, and Maria J. Serna. Approximating layout problems on random geometric graphs. *Journal of Algorithms*, 39(1):78–116, 2001.

[44] Josep Díaz, Jordi Petit, and Maria Serna. A survey of graph layout problems. *ACM Computing Surveys*, 34(3):313–356, 2002.

[45] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer Science & Business Media, 2012.

[46] Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

[47] Rodney G. Downey, Michael R. Fellows, and Ulrike Stege. Parameterized complexity: A framework for systematically confronting computational intractability. In Ronald L. Graham, Jan Kratochvíl, Jaroslav Nesetril, and Fred S. Roberts, editors, *Contemporary Trends in Discrete Mathematics: From DIMACS and DIMATIA to the Future, Proceedings of a DIMACS Workshop, Stir n Castle, Czech Republic, May 19-25, 1997*, volume 49 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 49–99. DIMACS/AMS, 1999.

[48] Shimon Even. NP-completeness of several arrangement problems. *Technical Report; Department of Computer Science, The Technion, Haifa, Israel*, 43, 1975.

[49] Michael R. Fellows, Danny Hermelin, Frances Rosamond, and Hadas Shachnai. Tractable parameterizations for the minimum linear arrangement problem. *ACM Transactions on Computation Theory*, 8(2):6, 2016.

[50] Michael R. Fellows, Bart M.P. Jansen, and Frances Rosamond. Towards fully multivariate algorithmics: Parameter ecology and the deconstruction of computational complexity. *European Journal of Combinatorics*, 34(3):541–566, 2013.

[51] Michael R. Fellows and Michael A. Langston. On well-partial-order theory and its application to combinatorial problems of VLSI design. *SIAM Journal of Discrete Mathematics*, 5(1):117–126, 1992.

[52] Michael R. Fellows, Daniel Lokshtanov, Neeldhara Misra, Frances A. Rosamond, and Saket Saurabh. Graph layout problems parameterized by vertex cover. In Seok-Hee Hong, Hiroshi Nagamochi, and Takuro Fukunaga, editors, *Algorithms and Computation, 19th International Symposium, ISAAC 2008, Gold Coast, Australia, December 15-17, 2008. Proceedings*, volume 5369 of *Lecture Notes in Computer Science*, pages 294–305. Springer, 2008.

[53] Fedor V. Fomin, Mathieu Liedloff, Pedro Montealegre, and Ioan Todinca. Algorithms parameterized by vertex cover and modular width, through potential maximal cliques. In R. Ravi and Inge Li Gørtz, editors, *Algorithm Theory - SWAT 2014 - 14th Scandinavian Symposium and Workshops, Copenhagen, Denmark, July 2-4, 2014. Proceedings*, volume 8503 of *Lecture Notes in Computer Science*, pages 182–193. Springer, 2014.

[54] Lance Fortnow. The status of the P versus NP problem. *Communications of the ACM*, 52(9):78–86, 2009.

[55] Delbert Fulkerson and Oliver Gross. Incidence matrices and interval graphs. *Paci c Journal of Mathematics*, 15(3):835–855, 1965.

[56] Jakub Gajarský, Michael Lampis, and Sebastian Ordyniak. Parameterized algorithms for modular-width. In Gregory Gutin and Stefan Szeider, editors, *Parameterized and Exact Computation - 8th International Symposium, IPEC 2013, Sophia Antipolis, France, September 4-6, 2013, Revised Selected Papers*, volume 8246 of *Lecture Notes in Computer Science*, pages 163–176. Springer, 2013.

[57] Robert Ganian. Twin-cover: Beyond vertex cover in parameterized algorithmics. In Dániel Marx and Peter Rossmanith, editors, *Parameterized and Exact Computation - 6th International Symposium, IPEC 2011, Saarbrücken, Germany, September 6-8, 2011. Revised Selected Papers*, volume 7112 of *Lecture Notes in Computer Science*, pages 259–271. Springer, 2011.

[58] Robert Ganian. Improving vertex cover as a graph parameter. *Discrete Mathematics & Theoretical Computer Science*, 17(2):77–100, 2015.

[59] Robert Ganian, Petr Hliněný, Jaroslav Nešetřil, Jan Obdržálek, Patrice Ossona de Mendez, and Reshma Ramadurai. When trees grow low: Shrubs and fast MSO1. In Branislav Rovan, Vladimiro Sassone, and Peter Widmayer, editors, *Mathematical Foundations of Computer Science 2012 - 37th International Symposium, MFCS 2012, Bratislava, Slovakia, August 27-31, 2012. Proceedings*, volume 7464 of *Lecture Notes in Computer Science*, pages 419–430. Springer, 2012.

[60] Frédéric Gardi. The Roberts characterization of proper and unit interval graphs. *Discrete Mathematics*, 307(22):2906–2908, 2007.

[61] Serge Gaspers, Margaret-Ellen Messinger, Richard J. Nowakowski, and Paweł Prałat. Clean the graph before you draw it! *Information Processing Letters*, 109(10):463–467, 2009.

[62] F. Gavril. Some NP-complete problems on graphs. In *Proceedings of the 11th Conference on Informational Science and Systems, 1977*, pages 91–95. Johns Hopkins University, 1977.

[63] Archontia C. Giannopoulou, Michal Pilipczuk, Jean-Florent Raymond, Dimitrios M. Thilikos, and Marcin Wrochna. Cutwidth: Obstructions and algorithmic aspects. *Algorithmica*, 81(2):557–588, 2019.

[64] M.A. Goldberg and I.A. Klipker. Minimal placing of trees on a line. In *Technical Report, Physico-Technical Institute of Low Temperatures*. Academy of Sciences of Ukranian SSR, USSR, 1976.

[65] Martin Charles Golumbic. Algorithmic Graph Theory and Perfect Graphs. In *Annals of Discrete Mathematics*, volume 57, pages 1–314. Elsevier, Amsterdam, The Netherlands, 2004.

[66] Jan Gorzny. On end vertices of search algorithms. Master's thesis, University of Victoria, Victoria, BC, Canada, 2015.

[67] Jan Gorzny. Computing imbalance-minimal orderings for bipartite permutation graphs and threshold graphs. In Weili Wu and Zhongnan Zhang, editors, *Combinatorial Optimization and Applications - 14th International Conference, COCOA 2020, Dallas, TX, USA, December 11-13, 2020, Proceedings*, volume 12577 of *Lecture Notes in Computer Science*, pages 766–779. Springer, 2020.

[68] Jan Gorzny and Jonathan F. Buss. Imbalance, cutwidth, and the structure of optimal orderings. In Ding-Zhu Du, Zhenhua Duan, and Cong Tian, editors, *Computing and Combinatorics - 25th International Conference, COCOON 2019, Xi'an, China, July 29-31, 2019, Proceedings*, volume 11653 of *Lecture Notes in Computer Science*, pages 219–231. Springer, 2019.

[69] Jan Gorzny and Jing Huang. End-vertices of LBFS of (AT-free) bigraphs. *Discrete Applied Mathematics*, 225:87–94, 2017.

[70] Jan Gorzny and Jing Huang. End-vertices of AT-free bigraphs. In Donghyun Kim, R.N. Uma, Zhipeng Cai, and Dong Hoon Lee, editors, *Computing and Combinatorics - 26th International Conference, COCOON 2020, Atlanta, GA, USA, August 29-31, 2020, Proceedings*, volume 12273 of *Lecture Notes in Computer Science*, pages 52–63. Springer, 2020.

[71] Jens Gramm, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. Data reduction and exact algorithms for clique cover. *Journal of Experimental Algorithmics*, 13:2, 2009.

[72] Peter L. Hammer and Stéphane Földes. Split graphs. *Congressus Numerantium*, 19:311–315, 1977.

[73] Maurice Hanan and Jerome M. Kurtzberg. A review of the placement and quadratic assignment problems. *SIAM Review*, 14(2):324–342, 1972.

[74] L. H. Harper. Optimal assignments of numbers to vertices. *Journal of the Society for Industrial and Applied Mathematics*, 12(1):131–135, 1964.

[75] Lawrence Hueston Harper. Optimal assignments of numbers to vertices. *Journal of the Society for Industrial and Applied Mathematics*, 12(1):131–135, 1964.

[76] Pinar Heggernes, Dieter Kratsch, and Daniel Meister. Bandwidth of bipartite permutation graphs in polynomial time. *Journal of Discrete Algorithms*, 7(4):533–544, 2009.

[77] Pinar Heggernes, Daniel Lokshtanov, Rodica Mihai, and Charis Papadopoulos. Cutwidth of split graphs and threshold graphs. *SIAM Journal on Discrete Mathematics*, 25(3):1418–1437, 2011.

[78] Pinar Heggernes, Daniel Meister, and Andrzej Proskurowski. Minimum distortion embeddings into a path of bipartite permutation and threshold graphs. In Joachim Gudmundsson, editor, *Algorithm Theory - SWAT 2008, 11th Scandinavian Workshop*

on Algorithm Theory, Gothenburg, Sweden, July 2-4, 2008, Proceedings, volume 5124 of *Lecture Notes in Computer Science*, pages 331–342. Springer, 2008.

[79] Pinar Heggernes, Pim van 't Hof, Daniel Lokshtanov, and Jesper Nederlof. Computing the cutwidth of bipartite permutation graphs in linear time. *SIAM Journal on Discrete Mathematics*, 26(3):1008–1021, 2012.

[80] Pavol Hell and Jing Huang. Interval bigraphs and circular arc graphs. *Journal of Graph Theory*, 46(4):313–327, 2004.

[81] Robert Hochberg, Colin McDiarmid, and Michael Saks. On the bandwidth of triangulated triangles. *Discrete Mathematics*, 138(1-3):261–265, 1995.

[82] M. A. Iordanskii. Minimal numberings of the vertices of trees—approximate approach. In *International Conference on Fundamentals of Computation Theory*, pages 214–217. Springer, 1987.

[83] Beverly Jamison and Stephan Olariu. On the semi-perfect elimination. *Advances in Applied Mathematics*, 9(3):364–376, 1988.

[84] Michael Jünger, Gerhard Reinelt, and Giovanni Rinaldi. The traveling salesman problem. *Handbooks in Operations Research and Management Science*, 7:225–330, 1995.

[85] Goos Kant. Drawing planar graphs using the canonical ordering. *Algorithmica*, 16(1):4–32, 1996.

[86] Goos Kant and Xin He. Regular edge labeling of 4-connected plane graphs and its applications in graph drawing problems. *Theoretical Computer Science*, 172(1-2):175–193, 1997.

[87] Jan Kára, Jan Kratochvil, and David R. Wood. On the complexity of the balanced vertex ordering problem. *Discrete Mathematics & Theoretical Computer Science*, 9:193–202, 2007.

[88] David R. Karger. A randomized fully polynomial time approximation scheme for the all-terminal network reliability problem. *SIAM Journal on Computing*, 29(2):492–514, 1999.

[89] Ton Kloks, Dieter Kratsch, and Haiko Müller. Bandwidth of chain graphs. *Information Processing Letters*, 68(6):313–315, 1998.

[90] Ephraim Korach and Nir Solel. Tree-width, path-width, and cutwidth. *Discrete Applied Mathematics*, 43(1):97–101, 1993.

[91] Dieter Kratsch. Finding the minimum bandwidth of an interval graph. *Information and Computation*, 74(2):140–158, 1987.

[92] Dieter Kratsch, Mathieu Liedloff, and Daniel Meister. End-vertices of graph search algorithms. In Vangelis Th. Paschos and Peter Widmayer, editors, *Algorithms and Complexity - 9th International Conference, CIAC 2015, Paris, France, May 20-22, 2015. Proceedings*, volume 9079 of *Lecture Notes in Computer Science*, pages 300–312. Springer, 2015.

[93] Matjaž Krnc and Nevena Pivač. Graphs where search methods are indistinguishable. In Jaroslav Nešetřil, Guillem Perarnau, Juanjo Rué, and Oriol Serra, editors, *Extended Abstracts EuroComb 2021*, pages 351–358, Cham, 2021. Springer International Publishing.

[94] Michael Lampis. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica*, 64(1):19–37, 2012.

[95] Frank Thomson Leighton and Satish Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM*, 46(6):787–832, 1999.

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

[97] Thomas Lengauer. Upper and lower bounds on the complexity of the min-cut linear arrangement problem on trees. *SIAM Journal on Algebraic Discrete Methods*, 3(1):99–113, 1982.

[98] Peng Li and Yaokun Wu. A four-sweep LBFS recognition algorithm for interval graphs. *Discrete Mathematics & Theoretical Computer Science*, 16(3):23–50, 2014.

[99] Simen Lilleeng. A polynomial-time solvable case for the NP-hard problem cutwidth. Master's thesis, University of Bergen, 2014.

[100] Lan Lin, Yixun Lin, and Douglas B. West. Cutwidth of triangular grids. *Discrete Mathematics*, 331:89–92, 2014.

[101] Yixun Lin, Xianglu Li, and Aifeng Yang. A degree sequence method for the cutwidth problem of graphs. *Applied Mathematics-A Journal of Chinese Universities*, 17(2):125–134, 2002.

[102] Jiuqiang Liu and Kenneth Williams. On bandwidth and edgesum for the composition of two graphs. *Discrete Mathematics*, 143(1-3):159–166, 1995.

[103] Daniel Lokshtanov. Parameterized integer quadratic programming: Variables and coefficients, 2015. arXiv preprint, https://arxiv.org/abs/1511.00310.

[104] Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Imbalance is fixed parameter tractable. *Information Processing Letters*, 113(19-21):714–718, 2013.

[105] Peter J. Looges and Stephan Olariu. Optimal greedy algorithms for indifference graphs. *Computers & Mathematics with Applications*, 25(7):15–25, 1993.

[106] Nadimpalli V.R. Mahadev and Uri N. Peled. Threshold Graphs and Related Topics. In *Annals of Discrete Mathematics*, volume 56, pages 1–543. Elsevier, Amsterdam, The Netherlands, 1995.

[107] Fillia Makedon, Christos H. Papadimitriou, and Ivan Hal Sudborough. Topological bandwidth. In Giorgio Ausiello and Marco Protasi, editors, *CAAP'83, Trees in Algebra and Programming, 8th Colloquium, L'Aquila, Italy, March 9-11, 1983, Proceedings*, volume 159 of *Lecture Notes in Computer Science*, pages 317–331. Springer, 1983.

[108] Fillia Makedon and Ivan Hal Sudborough. On minimizing width in linear layouts. *Discrete Applied Mathematics*, 23(3):243–265, 1989.

[109] Stephan Mertens. The easiest hard problem: Number partitioning. In Allon G. Percus, Gabriel Istrate, and Cristopher Moore, editors, *Computational Complexity and Statistical Physics*, Studies in the sciences of complexity, pages 125–140. Oxford University Press, 2006.

[110] George B. Mertzios, André Nichterlein, and Rolf Niedermeier. A linear-time algorithm for maximum-cardinality matching on cocomparability graphs. *SIAM Journal on Discrete Mathematics*, 32(4):2820–2835, 2018.

[111] Neeldhara Misra and Harshil Mittal. Imbalance parameterized by twin cover revisited. In Donghyun Kim, R. N. Uma, Zhipeng Cai, and Dong Hoon Lee, editors,

Computing and Combinatorics - 26th International Conference, COCOON 2020, Atlanta, GA, USA, August 29-31, 2020, Proceedings, volume 12273 of *Lecture Notes in Computer Science*, pages 162–173. Springer, 2020.

[112] Burkhard Monien. The bandwidth minimization problem for caterpillars with hair length 3 is NP-complete. *SIAM Journal on Algebraic Discrete Methods*, 7(4):505–512, 1986.

[113] Burkhard Monien and Ivan Hal Sudborough. Min cut is NP-complete for edge weighted trees. *Theoretical Computer Science*, 58(1-3):209–229, 1988.

[114] Petra Mutzel. A polyhedral approach to planar augmentation and related problems. In Paul G. Spirakis, editor, *Algorithms - ESA '95, Third Annual European Symposium, Corfu, Greece, September 25-27, 1995, Proceedings*, volume 979 of *Lecture Notes in Computer Science*, pages 494–507. Springer, 1995.

[115] Koji Nakano. Linear layout of generalized hypercubes. *International Journal of Foundations of Computer Science*, 14(1):137–156, 2003.

[116] Jan Obdržálek. *Graphs, Their Width, and Logic*. Habilitation thesis, Masaryk University, 2017.

[117] Achilleas Papakostas and Ioannis G. Tollis. Algorithms for area-efficient orthogonal drawings. *Computational Geometry*, 9(1-2):83–110, 1998.

[118] H.P. Patil. On the structure of k-trees. *Journal of Combinatorics, Information and System Sciences*, 11(2-4):57–64, 1986.

[119] Myriam Preissmann, Dominique de Werra, and Nadimpalli V.R. Mahadev. A note on superbrittle graphs. *Discret. Math.*, 61(2-3):259–267, 1986.

[120] Andrzej Proskurowski and Jan Arne Telle. Classes of graphs with restricted interval models. *Discrete Mathematics and Theoretical Computer Science*, 3(4):167–176, 1999.

[121] Satish Rao and Andréa W. Richa. New approximation techniques for some linear ordering problems. *SIAM Journal on Computing*, 34(2):388–404, 2004.

[122] André Raspaud, Ondrej Sỳkora, and Imrich Vrt'o. Cutwidth of the de bruijn graph. *RAIRO-Theoretical Informatics and Applications-Informatique Theorique et Applications*, 29(6):509–514, 1995.

[123] José D. P. Rolim, Ondrej Sýkora, and Imrich Vrto. Optimal cutwidths and bisection widths of 2- and 3-dimensional meshes. In Manfred Nagl, editor, *Graph-Theoretic Concepts in Computer Science, 21st International Workshop, WG '95, Aachen, Germany, June 20-22, 1995, Proceedings*, volume 1017 of *Lecture Notes in Computer Science*, pages 252–264. Springer, 1995.

[124] Donald J. Rose, Robert Endre Tarjan, and George S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on Computing*, 5(2):266–283, 1976.

[125] Róbert Sasák. Comparing 17 graph parameters. Master's thesis, University of Bergen, 2010.

[126] Farhad Shahrokhi, Ondrej Sýkora, László A. Székely, and Imrich Vrto. On bipartite drawings and the linear arrangement problem. *SIAM Journal on Computing*, 30(6):1773–1789, 2000.

[127] Yossi Shiloach. A minimum linear arrangement algorithm for undirected trees. *SIAM Journal on Computing*, 8(1):15–32, 1979.

[128] Michael Sipser. *Introduction to the Theory of Computation*. PWS Publishing, 1997.

[129] Jeremy Spinrad, Andreas Brandstädt, and Lorna Stewart. Bipartite permutation graphs. *Discrete Applied Mathematics*, 18(3):279–292, 1987.

[130] Robert Endre Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[131] Robert Endre Tarjan and Mihalis Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, 13(3):566–579, 1984.

[132] Robert Endre Tarjan and Mihalis Yannakakis. Addendum: Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, 14(1):254–255, 1985.

[133] Dimitrios M. Thilikos, Maria Serna, and Hans L. Bodlaender. Cutwidth I: A linear time fixed parameter algorithm. *Journal of Algorithms*, 56(1):1–24, 2005.

[134] Dimitrios M. Thilikos, Maria Serna, and Hans L. Bodlaender. Cutwidth II: Algorithms for partial w-trees of bounded degree. *Journal of Algorithms*, 56(1):25–49, 2005.

[135] Ryuhei Uehara. Bandwidth of bipartite permutation graphs. In Seok-Hee Hong, Hiroshi Nagamochi, and Takuro Fukunaga, editors, *Algorithms and Computation, 19th International Symposium, ISAAC 2008, Gold Coast, Australia, December 15-17, 2008. Proceedings*, volume 5369 of *Lecture Notes in Computer Science*, pages 824–835. Springer, 2008.

[136] Dong Wang, Edmund M. Clarke, Yunshan Zhu, and James H. Kukula. Using cutwidth to improve symbolic simulation and Boolean satisfiability. In *Proceedings of the Sixth IEEE International High-Level Design Validation and Test Workshop 2001, Monterey, California, USA, November 7-9, 2001*, pages 165–170. IEEE Computer Society, 2001.

[137] David R. Wood. Optimal three-dimensional orthogonal graph drawing in the general position model. *Theoretical Computer Science*, 299(1-3):151–178, 2003.

[138] David R. Wood. Minimising the number of bends and volume in 3-dimensional orthogonal graph drawings with a diagonal vertex layout. *Algorithmica*, 39(3):235–253, 2004.

[139] Mihalis Yannakakis. A polynomial algorithm for the min-cut linear arrangement of trees. *Journal of the ACM*, 32(4):950–988, 1985.

[140] Jinjiang Yuan and Sanming Zhou. Optimal labelling of unit interval graphs. *Applied Mathematics*, 10(3):337–344, 1995.

[141] Meibiao Zou, Zhifeng Wang, Jianxin Wang, and Yixin Cao. End vertices of graph searches on bipartite graphs. *Information Processing Letters*, 173:106176, 2022.

First Appearance of Symbols

$(I;k)$, 26
$(\ ^{A};\ ^{B})$, 24
D_t, 172
$E(V)$, 12
$E(X;Y)$, 12
$G[X]$, 10
I_S, 81
I_n, 10
K_n, 10
$K_{r;s}$, 16
L_i, 81
$L^{\cdot}(w)$, 12
M_k, 109
$N(X)$, 11
$N(v)$, 11
$N[X]$, 11
$N[v]$, 11
$N_G(v)$, 11
$N_G[v]$, 11
$N^{z}(a)$, 167
P_t, 11
$T^{\cdot}$, 108
$U(G)$, 12
$V(u;v)$, 119
$\Delta(G)$, 11
, 14
$>$, 14
, 14
$<\ X$, 14
$<$, 14
$\grave{}(v)$, 194
$\hat{}\ (X)$, 102
B, 125
$W(G)$, 5
$W(\ )$, 5
FPT, 26
NP, 25
P, 25
XP, 27
coNP, 25
$(\ )$, 77
(X), 15
(v), 15
pred (Y), 15
pred (v), 4
, 44
rank (v), 15
$_{x}$, 14
$_{>x}$, 14
$_{x}$, 14
$_{<x}$, 14
(i), 14
(v), 14
, 14
^{R}, 14
$_{<X}$, 14

$_{<i}$, 14
$_{>X}$, 14
$_{X}$, 14
$_{\ \ X}$, 14
$_{\ \ X}$, 14
succ (Y), 15
succ (v), 4
diam(G), 12
ecc$_G(w)$, 12
rank$(S; v)$, 81
$c\ (i)$, 14
$c\ (v)$, 3
$cc(G)$, 33
$cost(B)$, 125
$cost(e; B)$, 125
$cw(\ )$, 3
$cwd(G)$, 29
$d(v)$, 11
$d_G(u; v)$, 12
$d_G(v)$, 11
m, 10
$mw(G)$, 29
n, 10
$nd(G)$, 29
$pw(G)$, 27
$rtc(G)$, 37
$sd(G)$, 209
$tc(G)$, 29, 135
$tt(G)$, 77
$tw(G)$, 29
$vc(G)$, 27
$w\ (e)$, 5

Index

adjacency property, 24, 25, 186
admissible vertex, 23, 120, 164, 166, 167, 170
asteroidal triple, v, 17, 18, 22, 170, 207
AT-free
  bigraph, 23
  graph, v, 3, 7, 8, 22, 23, 53, 164, 167, 204–206, 211
avoids, 12, 166

backward, 15, 57, 59, 99
Bandwidth, 125, 148, 209
BFS, 46, 48, 163, 164, 170, 175, 206
  end-vertex, 166, 170–174, 176, 180
BFS-End-Vertex, 166, 170, 180, 181, 206, 211
bigraph, 16, 164
bipartite graph, 8, 16, 52, 92, 95, 131, 163, 167, 173, 201, 211
bipartite permutation graph, 7, 23, 24, 53, 136, 137, 139, 148, 166–171, 174–176, 181, 183, 185–187, 190, 191, 193–197, 202–204, 206, 209
bisection, 125
  cost of, 125, 127
  cost of an edge, 125
breadth-first search, v, 8, 46, 166
brittle graph, 19

C-good ordering, 131
C-good prefix ordering, 131
caterpillar graph, 148
chord
  of a cycle, 10
  of a path, 11
chordal graph, 3, 16–18, 22, 53, 121, 163, 194
chordless
  cycle, 10, 22
  path, 11
claw graph, 16, 17, 120, 121
clique, 10, 33, 69, 118, 120, 121, 136, 138, 139, 141, 147, 154
  maximal, 10
clique width, 29, 45, 55, 206
cocomparability graph, 148, 163
cograph, 53, 55
complete bipartite graph, 8, 16, 60, 111
complete graph, 8, 10, 60, 111, 118, 121, 154
complete join, 18
complexity class, 25
  FPT, 26, 93
  NP, 2, 25, 26, 52, 53, 92, 118, 126, 130, 147, 148, 164, 204, 208, 211
  P, 25, 26, 95, 170, 180, 193, 202, 204, 209

XP, 27, 93
coNP, 25, 26
coNP=poly, 25, 26, 130
component, 11, 46, 88, 89, 118, 160, 166, 168, 194, 198, 200, 201, 206
deep, 168, 169, 172, 174–176, 206
non-trivial, 11, 37, 88, 159–161, 168, 170, 195–197
trivial, 11, 161, 168, 195–197
connected, 11
consecutive, 14, 56, 69, 77, 80, 83, 96, 102, 113, 119, 139, 140, 143, 150, 154, 155, 159, 161, 187, 189
cover, 27
cut vertex, 11, 163, 194, 197, 201
Cutwidth, v, 3–5, 7–9, 52–56, 65, 66, 80–83, 88, 91–93, 95, 108, 112, 118, 119, 126, 130, 131, 134–136, 143, 144, 150, 156, 205–209
cutwidth, 1, 3, 14, 29, 127
of a complete bipartite graph, 60
of a complete graph, 60
of a graph, 3, 55, 65, 77
of a reverse ordering, 57
of an ordering, 3, 59, 72
of Möbius ladders, 109
of proper interval k-trees, 125
of superfragile graphs, 118
of threshold graphs, 143
of triangulated triangles, 108
relation to imbalance, 66
cycle, 10, 164

degree, 11, 57
depth-first search, v, 8, 48, 163
DFS, 48, 163, 164, 206
end-vertex, 186, 187, 190
DFS-End-Vertex, 166, 190, 191, 206
diameter, 12, 138, 139, 167, 183
diametrical
dominating pair, 166, 167, 170, 184
vertices, 12
distance, 12
Distortion, 209
dominates, 172
dominating pair, 166, 167, 172
dominating path, 185

eccentric vertex, 12, 168, 170, 171, 183–186
eccentricity, 12, 46, 167
edge clique cover, 33, 35, 80
edge clique cover number, 8, 33–35, 55, 80, 95, 205
edge-maximal graph, 121
elimination ordering, 19, 20
enclosure property, 24, 25, 186
end-vertex, 2, 6, 8, 120, 163, 164, 206
endpoint, 11
equivalence class of true twins, v, 7, 13, 77, 88, 102, 113, 119, 139, 140, 150, 154, 155, 161, 205
number of, 77, 113

false twins, 13, 58, 77
first-order logic, 43
fixed-parameter tractable, v, 8, 26, 80, 88, 91–93, 131, 155, 159, 205, 206
force of a vertex, 15
forward, 16, 57, 59, 99
FPT, 26, 27, 131, 205, 206, 208, 209
reducible, 26
reduction, 26, 130

generic search, 46, 48, 194, 201
end-vertex, 194
graph

definition, 10
induced, 10
Graph Cleaning, 93
grid graph, 148

Hamiltonian cycle, 189, 190
Hamiltonian path, 186–190
hit, 12, 166
(House, Hole, Domino)-free graph, 164
hypercube graph, 148
hyperedge, 125
hypergraph, 125, 126
Hypergraph Minimum Bisection, 125–127, 130

ILP, 81
Imbalance, v, 3–5, 7–9, 55, 65, 66, 92–95, 108, 118, 125, 127, 130, 131, 134–136, 138, 141, 142, 146, 147, 150, 154, 205–209
imbalance, 3, 60
of k-trees, 119
of a graph, 4, 55, 65, 111
of a reverse ordering, 95
of a vertex, 4, 95, 97, 98
of an ordering, 4, 15, 99, 100, 102
of complete bipartite graph, 111
of complete graph, 111
of Möbius ladders, 111
of proper interval k-trees, 125
of superfragile graphs, 118
of triangulated triangles, 108
relation to cutwidth, 66
independent set, 10, 40, 81, 93, 112, 131, 141, 156, 167
induced subgraph, 10
integer linear program, 81–83, 89, 91, 135, 209
Integer Quadratic Programming, 155
integer quadratic program, 150, 155, 160, 161
intercept, 12, 166
interval graph, 17, 18, 22, 53, 95, 118, 120, 148, 164, 204, 206–208
inverted pair, 143, 144
inverted-clique-pair, 142, 143
inverted-independent-pair, 142
IQP, 155
isolated vertex, 11

k-tree, 17, 119, 121, 125
kernel, 8, 26, 53, 93, 125, 130, 206, 208
size of, 27
kernelization algorithm, 26

label of a vertex, 194
layer, 12, 46, 167, 196
layer search, 46, 166, 168, 170, 206
LBFS, 48, 120, 164, 167, 170, 183, 186, 206
end-vertex, 120, 163, 164, 166, 170, 171, 183, 185
LBFS$^+$, 48, 119, 120
LBFS-End-Vertex, 164, 170, 183, 185, 186, 206
LDFS, 50, 163, 164, 207, 209
length
of a cycle, 10
of a path, 11
lexicographic breadth-first search, v, 8, 48, 163
lexicographic depth-first search, 50, 163
linear arrangement, 1, 14
linear evaluation function, 43
linear layout, 1, 14

LinEMSO$_1$, 43, 45
location, 81, 83, 88, 156, 160, 161

Möbius ladder, 8, 109
matching, 163
maximal neighbourhood search, v, 8, 48, 166
maximum cardinality search, 50, 163
maximum degree, 11, 55, 92, 209
MCS, 50, 163, 164, 207, 209
Median Placement, 143
mesh graph, 53
midpoint, 11
MinCut, 143
Minimum Linear Arrangement, 5
Minimum Restricted Twin Cover, 44
Minimum Twin Cover, 44
Minimum Vertex Cover, 44
misses, 12, 22, 23, 166
MNS, 48, 164, 193–198, 200–202, 206
  end-vertex, 193, 194, 197, 201, 202
  ordering, 198
MNS-End-Vertex, 166, 193, 194, 202, 203, 206, 211
models, 43
  model checking, 43
modular width, 29
module, 29
  trivial, 29
monadic second-order logic, 41
MSO$_1$, 41, 43–45
MSO$_1$ Model Checking, 43
multihypergraph, 125

neighbourhood, 11
  closed, 11
  of a set, 11
  open, 11
neighbourhood diversity, 29, 34, 37, 39, 40, 92, 93, 95, 209
Number Partitioning, 118, 119

OLA, 4
Optimal Linear Arrangement, v, 3–9, 15, 119, 125, 148–150, 154–156, 159, 204, 205, 207–209
optimal linear arrangement, 4
  of superfragile graphs, 154
ordering, v, 1–3, 5, 6, 14, 15, 46, 56, 59, 65–67, 72, 96–98, 161
  consecutive set, 14
  regular, 121, 123, 125
  reverse, 14, 57, 95

p-Opt-ILP, 80, 81
p-Variable Integer Linear Programming Optimization, 80
parameterized problem, 26, 27
path, 11, 204
  dominating, 166
path decomposition, 29, 121
pathwidth, 27, 40, 53, 55, 121, 123
pendant, 11, 100, 147, 189
PEO, 163
perfect elimination ordering, 163
perfect graph, 23
perfectly balanced, 15, 93
permutation graph, 23, 148, 173
planar graph, 52, 92
polynomial parameter reducible, 130
predecessors, 4, 15
proper interval *k*-tree, 119, 121, 123, 125, 204, 205
proper interval bigraph, 23, 136

proper interval bipartite graph, 23
proper interval graph, 7, 17, 53, 119, 120, 123, 125, 136–139, 148

rank of a vertex, 15, 56–58, 81, 82, 97, 98, 100
  in a reverse ordering, 57
regular labelling, 119
regular ordering, 119, 120, 138–140
restricted twin cover, 37, 136
restricted twin cover number, 8, 37, 40, 41, 55, 80, 88, 95, 135, 150, 155, 159, 206

S-Beginning-End-Vertex, 7
S-End-Vertex, v, 6–9, 46, 164, 165, 209, 211
search algorithm, 2, 6, 7, 46, 163, 164, 206
separator, 11
  $(A; B)$, 11
  clique, 11
set of true twins, 13
set of twins, 13, 15, 58, 96
shrub depth, 209
simplicial vertex, 11, 120, 137–139, 163, 164
split graph, 17, 53, 147, 164, 172, 173
split partition, 17, 141
star, 16, 207
strong ordering, 24, 25, 186, 187
subdivide an edge, 12
substar, 194–197, 201, 202
successors, 4, 15
superbrittle graph, 19
superfragile graph, 7, 17–19, 53, 95, 112, 118, 135, 150, 154, 204, 207

threshold graph, 3, 17, 53, 141–145, 205, 209
threshold partition, 17, 141, 142, 145
to the left, 15
to the right, 16
topological bandwidth, 55
tree, 16, 53, 92, 148, 207
tree decomposition, 29
treewidth, 29, 53, 55, 65, 92, 209
triangulated graph, 16
triangulated triangle, 8, 108
trivially perfect graph, 207
true twins, v, 5, 7, 13, 29, 55, 58, 60, 77, 79, 80, 83, 93, 102, 113, 114, 139, 140, 150, 154, 161, 205–207
twin cover, 29, 44, 88, 89, 93, 135, 136, 159, 160, 206
twin cover number, 8, 37, 92, 93, 134, 135, 209
twins, 13
  false, 13
  true, 13
Type I problem, 2, 3, 14, 204, 205, 207, 209
Type II problem, 3, 6, 46, 206, 207

unit interval graph, 17
unit interval ordering, 119
universal, 12
universal vertices, 12, 20, 118, 138, 150, 153, 154
universe, 125, 130
unrelated vertices, 23, 120, 164

vertex cover, 27, 29, 44, 88, 89, 125–127, 131, 135, 159, 207
vertex cover number, 8, 27, 33, 34, 40, 53, 55, 80, 81, 89, 92, 93, 95, 135, 150, 155, 160, 161, 205, 206

vertex ordering characterization, 198
visits, 46
weakly chordal graph, 164, 166
weight
of a complete graph, 154
of a graph, 5, 150, 154
of an edge, 4, 5, 148, 150–152
of an ordering, 5