On Some Covering, Partition, and Connectivity Problems in Graphs

A dissertation submitted towards the degree Doctor of Natural Science of the Faculty of Mathematics and Computer Science of Saarland University

by Davis Issac

Saarbrücken / 2019

Day of Colloquium:	09. September 2019
Dean of the Faculty:	Prof. Dr. Sebastian Hack
Chair of the Committee:	
Reporters	
First reviewer:	Dr. Andreas Karrenbauer
Second reviewer:	Prof. Dr. Kurt Mehlhorn
Third reviewer:	Prof. Dr. L. Sunil Chandran
Academic Assistant:	Dr. Antonios Antoniadis

Dedicated to the loving memory of my Grandmother Annamma George.

Abstract

Abstract We look at some graph problems related to *covering*, *partition*, and *connectivity*. First, we study the problems of covering and partitioning edges with *bicliques*, especially from the viewpoint of parameterized complexity. For the partition problem, we develop much more efficient algorithms than the ones previously known. In contrast, for the cover problem, our lower bounds show that the known algorithms are probably optimal. Next, we move on to graph coloring, which is probably the most extensively studied partition problem in graphs. *Hadwiger's conjecture* is a long-standing open problem related to vertex coloring. We prove the conjecture for a special class of graphs, namely squares of 2-trees, and show that square graphs are important in connection with Hadwiger's conjecture. Then, we study a coloring problem that has been emerging recently, called *rainbow coloring.* This problem lies in the intersection of coloring and connectivity. We study different variants of rainbow coloring and present bounds and complexity results on them. Finally, we move on to another parameter related to connectivity called *spanning tree congestion* (STC). We give tight bounds for STC in general graphs and random graphs. While proving the results on STC, we also make some contributions to the related area of *connected partitioning.*

Zusammenfassung Wir betrachten einige Graphprobleme mit Bezug auf *Abdeckung*, *Partition* und *Konnektivität.* Zunächst untersuchen wir Kantenabdeckung und -partition mit *Bicliquen*, insbesondere im Hinblick auf parametrisierte Komplexität. Für das Partitionierungsproblem entwickeln wir sehr viel effizientere Algorithmen als die bisher bekannten. Für das Abdeckungsproblem hingegen zeigen unsere unteren Schranken, dass die bereits bekannten Algorithmen wahrscheinlich optimal sind. Als Nächstes betrachten wir *Graphfärbung*, das wahrscheinlich am meisten untersuchte Partitionsproblem in Graphen. *Hadwigers Vermutung* ist ein seit Langem bestehendes offenes Problem bezüglich der Knotenfärbung. Wir beweisen diese Vermutung für eine spezielle Graphklasse, nämlich Quadrate von 2-Bäumen, und zeigen, dass Quadratgraphen wichtig in Bezug auf Hadwigers Vermutung sind. Danach untersuchen wir das Problem der sogenannten *Regenbogenfärbung.* Dieses erst kürzlich entstandene Problem liegt im Schnittpunkt der Probleme Färbung und Konnektivität. Wir untersuchen verschiedene Varianten der Regenbogenfärbung und zeigen Schranken und Komplexitätsergebnisse. Schließlich gehen wir über zu einem weiteren Konnektivitätsparameter, der sogenannten *spanning tree congestion* (STC). Wir präsentieren scharfe Schranken für STC in allgemeinen und zufällig generierten Graphen. Unsere Ergebnisse in diesem Bereich leisten darüberhinaus einen Beitrag zu dem verwandten Gebiet der *verbundenen Partitionierung.*

Acknowledgments

First of all I would like to thank my advisor Andreas Karrenbauer. I thank him for accepting me as a PhD student and guiding me through my PhD for the past 4 years. He has always given me the freedom to follow my own research interests, at the same time making sure that I don't go much out of the track. I thank him for his contributions, technical and otherwise to my thesis.

I was very lucky to have Prof. Sunil Chandran visiting MPI-Informatics during my PhD term for about 18 months. He has been a co-author in most of my publications and has had a big effect on my shaping my research outlook. It has been a great pleasure to collaborate with him.

I thank Erik Jan for all the technical input and for the encouragement he has given me. I thank Prof. Ragesh Jaiswal, who was my Masters supervisor for being a mentor at the start of my research career. I thank all my other co-authors Anita, Marco, Juho, Paloma, Pinar, and Sanming for their contributions. I thank Kurt Mehlhorn, for giving me the opportunity to be a PhD student at the Algorithms department, and also for building such an encouraging atmosphere in the group. I thank Pavel, for being the most friendly and helpful officemate for 4 years. I think I have learned a lot from Pavel. I also thank the other current and past members of D1: Philip, Syamantak, Daniel, Andreas, Max, Ruben, Bojana, Michael Dirnberger, Sandy, Paresh, Parinya, Karl, Marvin, Arijit, Kunal, Attila, Bundit, Saeed, Bhaskar, Gorav, Anurag and others for their words of encouragement, helpful discussions, friendly chats etc. I thank Christina and Ingrid for helping with all the administrative stuff.

I thank my loving wife Treesa for all the emotional support, care and encouragement that she has given me. I thank my parents Issac and Lissy, and my siblings Louis, Anies and Agnus for their constant care and support for me. I also thank Treesa's parents for their well wishes. I thank my grandparents Mariyakkutty Issac, Annamma George and Edappally George for their love. I thank all my friends in Saarbruecken, especially Vijay, Saranya, Jacob, Chaitin, Sreekesh, Balu, Rohini, Roshna, Udaykumar, Thyagu, Sourav and others for the good times I had with them.

Contents

CHAPTER 1
Introduction

Graphs are simple and elegant structures that can abstract many real world problems. Consider the Facebook network; we say that every person is a *vertex*, and, if two people are friends, they are said to be connected by an *edge*. One might be interested in finding a subset of the largest number of people who are all pairwise friends with each other. In graph theory, we call this the *maximum clique* problem. Similarly, there are numerous fields of science and engineering, where the structures and problems arising can easily be modelled into graph structures and problems. Examples include finding shortest paths in traffic networks, community detection in social networks, analyzing structures of atoms and molecules, tracking spread of disease or parasites in species etc.

Graph theory, over the years, has supplied various tools and techniques for analyzing such structures and finding solutions to such problems. Over the past few decades, graph theory has developed into a separate branch of mathematics, and as an integral part of theoretical computer science. As a data structure, graphs are indispensable to any computer scientist. Moreover, graph theory is an appealing self-contained theory, which is worth studying just for its combinatorial beauty.

In this thesis, we look at some graph problems, some having practical relevance, and some of theoretical importance. Some of the problems are *algorithmic* in nature, whereas some are *structural* in nature. The main themes of the thesis are *covering*, *partition*, and *connectivity* in graphs.

In covering/partition problems, we want to cover/partition a graph using subgraphs having a particular structural property. One such subgraph that we are interested in is a *biclique*, also known as a *complete bipartite graph*. We will examine covering and partitioning graphs with bicliques from an algorithmic viewpoint. In particular, the focus will be on the parameterized complexity aspects of biclique cover/partition and related problems. Bicliques are inherently related to matrix factorization and matrix rank, and hence, we will also discuss some related matrix problems.

Some of the most extensively studied partition problems in graphs includes different kinds of *coloring* problems. In coloring problems, we want to color vertices/edges such that each color class induces a graph with some specific structural property. The most intensively studied coloring problem has been the vertex coloring problem, which is basically partitioning the graph into *independent sets*. An independent set is nothing but a set of vertices that are all pairwise non-adjacent. Some of the deepest theorems and conjectures in graph theory are related to vertex coloring. We will look at one such conjecture called the *Hadwiger's Conjecture*, and show some interesting results related to it.

Next, we move onto a graph problem called rainbow coloring, which combines coloring with connectivity. We study the graph theoretic and algorithmic aspects of rainbow coloring and some of its variants.

Connectivity is one of the most important topics in graph theory. Various connectivity problems, such as hamiltonicity and 2-connectivity, have been dealt with extensively in the literature. An important structural object related to connectivity is a *spanning tree* of the graph. A parameter of the spanning tree that is important in many network applications, is its edge-congestion. We examine the congestion of spanning trees, and also the related area of *connected partitioning*. Connected partitioning refers to partitioning the graph such that, each of the parts, is connected, and obeys some specified cardinality/weight constraints.

We consider both structural and algorithmic aspects in the above areas. Most of the algorithmic problems we encounter are NP-hard. Being NP-hard implies that these problems are unlikely to admit an algorithm that is efficient, precise, and that work for all instances. Some ways to tackle such problems are to aim for an approximate solution, or to study the problem in specific input instances, or to develop algorithms whose running times are parameterized by some parameter which we expect to be small in many relevant instances. The techniques required for these methods have been developed in the fields of approximation algorithms, theory of graph classes, and parameterized algorithms. We use techniques from these fields to address the algorithmic problems.

1.1 Biclique Cover and Partition

The first problems that we deal with are the biclique cover and partition problems. Bicliques, also known as complete bipartite graphs, are interesting graph structures, which appear in many real-world applications. In the biclique cover/partition problem, we want to cover/partition the edges of a graph with as few bicliques as possible. The number of bicliques required is called *biclique cover/partition number*. We will be most interested in the case when the input graph is a bipartite graph, as this has parallels in the area of matrix factorization. More specifically, the biclique cover and partition problems in bipartite graphs correspond to finding binary and boolean decompositions of matrices. The biclique cover and partition problems have many real-world applications, some of which we discuss below.

1.1.1 Applications

Display optimization. One of the applications of biclique cover and partition is in display optimization. In some type of display monitors known as *passive matrix displays*, there is no dedicated switch for each pixel of the monitor. Instead, a switch is responsible for a whole row or a whole column of the display matrix. For example, if row i and column j is switched on, then pixel (i, j) glows. Because of this row-wide and column-wide addressing of the switches, an arbitrary shaped image cannot always be displayed in a single frame. A single frame can only display a rectangular (need not be a continuous rectangle) shaped sub-image. It is desirable to minimize the number of frames, as this helps in reducing the power used by the display. This problem of minimizing frames can be translated into the problem of finding a minimum biclique partition in a bipartite graph, as follows. We can map the display matrix of the monitor to a bipartite graph by taking each row as a left vertex and each column as a right vertex. The edges of the graph depends on the image to be displayed. Let us say, we want to display a black and white image. We will put an edge from the i-th vertex on the left side, to the j-th vertex on the right side if and only if the pixel (i, j) is white in the image. We call this graph, the *image graph*. In fact, any sub-image that can be displayed in a single frame corresponds to some biclique of the image graph. Hence, in order to display the image, we require a biclique partition of the image graph, and each partition is then addressed in a separate frame. Thus, in order to minimize the number of frames used, it is sufficient to find a biclique partition of minimum size of the image graph. The requirement is biclique partition and not biclique cover because, each white pixel in the image should be addressed by only one of the frames, in order to obtain an uniform illumination. In some displays, each pixel is allocated a memory, in which case multiple frames are allowed to address the same pixel. For such displays, one do not need a biclique partition, but a biclique cover is sufficient. In that case, the corresponding optimization problem is to find a minimum biclique cover of the image graph.

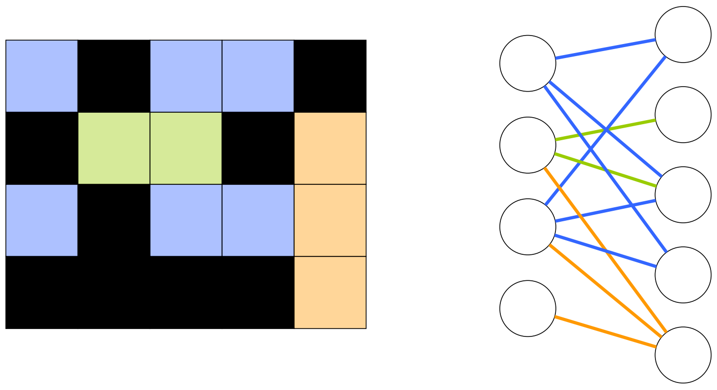


Figure 1.1: A representation of a black and white image as a biclique partition. The white pixels are given colors according to the corresponding bicliques.

Communication complexity. The biclique cover number and biclique partition number of bipartite graphs are related to the *communication complexity* in 2-party communication [89, 101]. The usual setting in 2-party communication is as follows. There is a function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$; Alice is given the first n bits of input and Bob is given the second n bits. So, Alice and Bob need to communicate to find the output of the function. The communication is carried out according to some protocol. The cost of a protocol is the maximum of the number of bits required to be communicated over all possible pair of inputs. The communication complexity of f is the minimum cost over all protocols computing f. There are 2 types of communication complexity depending on whether randomness is allowed in the protocol. In deterministic communication complexity, the protocol is constrained to be deterministic, and in non-deterministic communication complexity, the protocol is allowed to be randomized (in which case the protocol needs to compute f successfully with high probability). Now, we describe the connection to biclique cover and partition. The function f can be represented as a binary truth table with 2^n rows and 2^n columns. Hence, it can be easily interpreted as a bipartite graph, with rows as the left-side vertices and columns as the right-side vertices.

The logarithm (to base 2) of the biclique cover number of this graph gives exactly the non-deterministic communication complexity of f, and the logarithm (to base 2) of the biclique partition number gives a lower bound on the deterministic communication complexity.

Data analytics, data mining, data compression, and clustering. Finding small biclique cover/partitions of bipartite graphs is equivalent to finding matrix decompositions having low binary/boolean rank. In (approximate) low-rank decompositions, we represent (an approximation of) a given $m \times n$ matrix as the product (over an appropriate arithmetic, for e.g., binary or boolean arithmetic) of an $m \times k$ matrix and a $k \times n$ matrix, for some small k. Low-rank decompositions are of particular interest in data analytics, data mining, data compression and clustering. Since much of the real-world data has a binary or boolean nature, binary/boolean decompositions can be helpful in discovering and analyzing the semantics of the data. Low-rank decomposition of matrices clearly helps in compression, as instead of mn entries, we need to store only $(m+n)k$ entries for some small k. Low-rank decomposition into binary matrices is useful because, for a lot of real-world binary data, it is desirable to preserve the binary form in the compressed format in order to preserve some features represented by the data. Approximate matrix decompositions can also be interpreted as clustering with rank-constraints on the cluster-centers. Hence, approximate low-rank decompositions have applications in many clustering problems. See [19], for example.

Applications of biclique cover and partition can also be found in the areas of bioinformatics [128, 129], computer security [59], database tiling [73], finite automata [78], and graph drawing [60].

1.1.2 Background

Both biclique cover and partition problems are NP-hard and already appeared in the book by Garey and Johnson [70]. They also turn out to be very hard to approximate within reasonably good approximation factors [32].

Hence, we turn to another method of attacking NP-hard problems, that is, from the parameterized complexity point of view. In many applications, we have that the cover/partition size is usually much smaller in comparison to the input size, especially after some preprocessing steps. Hence, we study these problems parameterized by the size k of the cover/partition. In the matrix world, the parameter corresponds to the rank of the output matrix. It makes sense to parameterize by the rank because, in binary/boolean factorization, much of the focus is on obtaining *low-rank* decompositions.

Both biclique cover and partition problems were already shown to be fixed parameter tractable. However, there was a lack of reasonably efficient FPT algorithms, as the best known FPT algorithms were doubly exponential in k. Hence, we focus on developing more efficient FPT algorithms for the biclique cover and partition problems.

One of the important techniques in parameterized complexity, is called *kernelization*. Kernelization algorithms are, in essence, preprocessing algorithms. Preprocessing is important in connection with biclique cover and partition. In many of the real-world implementations of algorithms for these problems, the end-computation is done by some generic ILP solver, like Gurobi or CPLEX. Hence, it becomes very beneficial to do as much preprocessing as possible, that can reduce the size of the instance, before feeding it to the generic solver. Although, the generic solvers have preprocessing mechanisms of their own, most often they are unable to make use of preprocessing steps based on structural properties. The techniques in kernelization, usually exploit some structural properties to give simple and efficient preprocessing rules. Thus, it is worthwhile to study about the kernelization of the biclique cover/partition problems.

1.1.3 Our Contribution

We come up with a simple algorithm for biclique partition that runs in $\mathcal{O}^*(2^{\mathcal{O}(k^2)})$-time. This is a drastic improvement from the previous best known running time of $\mathcal{O}^*(2^{2^k})$. We achieve this result by exploiting the binary rank interpretation of the biclique partition problem in the matrix world and applying some linear algebraic techniques over it. The main technique that we develop is simple, employs only basic linear algebraic notions, and may turn out to be useful for other related problems. The core idea is that, we do not need to enumerate all the rows (or columns) of our target matrix decomposition, but rather only a set of few rows that can span all the other rows.

For the biclique cover problem, we were not successful in finding better FPT algorithms. Instead, we found that there are fundamental obstructions to getting such improvements. We demonstrate this by proving a hardness result for biclique cover. More specifically, we show that an improvement over the doubly exponential algorithm is not possible for biclique cover, unless the ETH is false. We also prove kernelization lower bounds for biclique cover, implying that the problem does not admit efficient preprocessing rules.

Our techniques also lead to improved results for some related problems such as edge clique partition and low-rank matrix approximation. We discuss the connections with these problems and show that our results imply novel results for them as well. We also demonstrate that the known kernelization algorithms for some of the cover/partition problems can be turned into polynomial-time approximation algorithms, which give better approximation ratios than the previously known algorithms.

1.2 Hadwiger's Conjecture for Squares of 2-Trees

The most extensively studied partition problem in graphs has to be the vertex coloring problem. In this problem, the goal is to partition the vertices into as few *independent sets* as possible. Some of the deepest theorems in graph theory, such as the four color theorem and the perfect graph theorem are related to vertex coloring. Hadwiger's conjecture is a far reaching generalization of the well-known *four color theorem.* It was proposed by Hugo Hadwiger in 1943 [80] and is as follows.

Conjecture (Hadwiger's Conjecture)**.** *Any K_{t+1}-minor free graph is t-colorable.*

Hadwiger's conjecture is known to be a challenging problem. Bollobás, Catlin, and Erdős [26] describe it as "one of the deepest unsolved problems in graph theory". Even after years of research and attempts by leading researchers in graph theory, the problem has seen very slow progress. For general graphs, the conjecture has been proved only for $t = 1, 2, \ldots, 5$ so far.

Similar to other difficult conjectures in graph theory, Hadwiger's conjecture has been attempted in many special classes of graphs. The motivation is that proving the conjecture for some natural graph class may lead to new techniques and reveal some structure for the general case. So far Hadwiger's conjecture has been proved for several classes of graphs, including line graphs [138], proper circular arc graphs [20], quasi-line graphs [49], 3-arc graphs [162], complements of Kneser graphs [163], and powers of cycles and their complements [106].

1.2.1 Our Contribution

We continue the research on Hadwiger's conjecture on special graph classes. We pursue a direction started by Reed and Seymour [138] and followed up by Chudnovsky and Fradkin [49]. Reed and Seymour [138] proved Hadwiger's conjecture for line graphs. Chudnovsky and Fradkin [49] proved Hadwiger's conjecture for quasi-line graphs, which are a generalization of line graphs. A quasi-line graph is a graph for which the neighborhood of every vertex can be written as the union of 2 cliques. We were interested in investigating Hadwiger's conjecture for further generalizations of quasi-line graphs. A natural generalization is d-groupable graphs, which are the graphs for which the neighborhood of every vertex can be written as a union of d many cliques. Since this appeared to be very hard, we turned our attention to a subclass, namely square graphs of degree-d bounded graphs, where a square of a graph G is defined as the graph whose vertex set is same as that of G and the edge set is the set of all pair of vertices that are at a distance (counted in number of edges) of at most 2 in G. But we found that proving the conjecture for square graphs has some natural obstructions. We demonstrate this by showing that Hadwiger's conjecture for squares of split graphs is as hard as Hadwiger's conjecture for general graphs. A split graph is a graph that can be decomposed into a clique and an independent set.

Theorem. *Hadwiger's conjecture is true for squares of split graphs if and only if it is true for general graphs.*

The above result, aroused our interest in investigating Hadwiger's conjecture for square graphs. At first glance, it appears that the above result may not make Hadwiger's conjecture any easier. Nevertheless, being able to restrict oneself to square graphs, may prove advantageous. Proving Hadwiger's conjecture in the square graphs of some special classes of graphs may reveal properties useful for the general case.

A k-tree is a graph that can be obtained by starting with a k-clique and applying the following step any number of times: We pick a clique of size k and extend it to a clique of size $k + 1$ by introducing a new vertex. The class of k-trees are a subclass of chordal graphs. The only difference in the construction of a chordal graph is that there is no restriction on the size of clique picked; we still extend it by one more vertex. Chordal graphs are known to be a superclass of split graphs and hence from the above theorem, we know that Hadwiger's conjecture for squares of chordal graphs is as hard as the general case. Hence, it is an interesting question whether there is a value of k, above which Hadwiger's conjecture on squares of k-trees becomes very hard to prove? If so, this might reveal some structural difficulties with respect to Hadwiger's conjecture. We kick-start research in this direction by proving the following theorem.

Theorem. *Hadwiger's conjecture is true for 2-trees.*

Our proof of Hadwiger's conjecture for squares of 2-trees can also be considered as progress in the direction of generalizations of line graphs started by Chudnovsky and Fradkin [49], in the following sense. The class of 2-*simplicial graphs* generalizes the class of quasi-line graphs studied by Chudnovsky and Fradkin [49]. A 2-simplicial graph is a graph for which there is an ordering of the vertices such that the higher numbered neighbors can be written as the union of 2 cliques. It is easy to observe that squares of 2-trees are a subclass of 2-simplicial graphs. We also know that squares of 2-trees are not completely contained in quasi-line graphs. Thus we have proved Hadwiger's conjecture for a subclass of 2-simplicial graphs that is not contained in quasi-line graphs.

1.3 Rainbow Coloring and its Variants

Rainbow coloring is a recently emerging area that lies at the intersection of coloring and connectivity, which are two of the main topics studied in this thesis. The concept of rainbow connectivity was first introduced by Chartrand et al. [43]. Rainbow colorings are special types of edge colorings with some connectivity constraints with respect to the coloring. A path in an edge colored graph is called a *rainbow path* if the edges of the path have pairwise distinct colors. An edge colored graph is said to be rainbow connected if between every pair of vertices there is a rainbow path. A coloring that makes the graph rainbow connected is called a rainbow coloring. The minimum number of colors needed for any rainbow coloring of the graph is called its *rainbow connection number* (rc).

Rainbow coloring has applications in communication networks. Chakraborty et al. [31] describes the following application: In a network, we want to establish message paths between every pair of vertices with the requirement that links in any such path should have distinct channels. And it is desirable to use as few channels as possible. This clearly maps to the rainbow coloring problem.

Rainbow coloring and its variants have become an important topic in graph theory with over 300 papers written on it (e.g. see the survey by Li et al. [113]). The interest is probably due to the rich combinatorial structures associated with the problem, and the fact that it interconnects coloring and connectivity, two of the most important topics in graph theory.

Chartrand et al. [43] also introduced a related concept called the *strong rainbow connectivity.* Here, the difference is that between every pair of vertices, we require a shortest path that is also a rainbow path. Compared to rainbow coloring, the strong version has been less studied, and there are not many bounds known for the strong rainbow connection number (src).

Krivelevich et al. [100] defined a natural variant of rainbow coloring and strong rainbow coloring in vertex colored graphs. Here, the constraint is that instead of edges, the internal vertices of the path are to be distinctly colored.

1.3.1 Our Contribution

Much of the research on rainbow coloring has focused on determining upper bounds for the rainbow connection number. It is known that $\mathsf{rc}(G) \leq n-1$. This can be seen by observing that coloring the edges of a spanning tree with distinct colors makes a graph rainbow connected. It is conjectured that $\mathsf{src}(G) \leq n-1$ and also stronger conjectures like $\mathsf{src}(G) \leq n+1-\chi(G)$ and $\mathsf{src}(G) \leq f(G)-1$ have been made [102], where $\chi(G)$ is the chromatic number and $f(G)$ is the size of the largest induced forest in G. For the last two conjectures, the first step would be to prove them for $\mathsf{rc}(G)$. We obtain partial progress for the above conjectures: We show that $\mathsf{rc}(G) \leq f(G)+2$ and also that $\mathsf{src}(G) \leq n+1-\chi(G)$ for chordal graphs.

In general, there are much fewer bounds known for src compared to rc, especially in structured graph classes. Hence, we decided to investigate further on src. We observed that many of the strong rainbow colorings that we discovered, have the additional property that every shortest path in the graph is a rainbow path. This led us to define such a restricted version of strong rainbow coloring called *very strong rainbow coloring*. We give many upper bounds for vsrc, both in general graphs and in special graph classes. Some of these bounds also imply previously unknown bounds for src. Further, we show some preliminary complexity results on the problem of computing the vsrc of a graph.

For the vertex variants, the complexity of computing was not that well-studied. Only a few basic results were known. We study the vertex variants in the context of two special graph classes, bipartite graphs and split graphs. We mainly show NP-completeness and hardness of approximation in these classes. We contrast them with positive results for some restricted classes such as bipartite permutation graphs, block graphs, and unit interval graphs.

1.4 Spanning Tree Congestion and Connected Partitioning

In this part of the thesis, we will deal with some connectivity related problems in graphs. The most fundamental graph structure related to connectivity is a *spanning tree* of the graph. A spanning tree can also be used to approximately represent a graph with very few edges, or in other words, *graph sparsification*. In flow networks such as transport networks, an important parameter for any sparsification is its *edge-congestion*. The edge-congestion of an edge in the sparsified graph is defined as the number of edges of the original graph that is routed through that edge. Hence, it is natural to seek a spanning tree of the graph, whose edges have small edge-congestion.

Graph sparsification problems, in general, are used to represent a graph with a smaller/sparser graph. In a communication network, a spanning tree of the network is a minimal subgraph that needs to be maintained so that all pairs of nodes still have a communication channel in the network. But, any single link should not be overloaded with too much traffic. Thus, finding spanning trees with small maximum congestion, where the maximum is taken over all the tree edges, finds applications in network design problems. The problem also finds applications in the areas of parallel computing and circuit design. The probabilistic version of the problem, called as *probabilistic capacity mapping*, where one is allowed to have a distribution of spanning trees instead of one spanning tree, finds applications in several important algorithmic graph problems, for example, the Min-Bisection problem [63].

Another quantity related to congestion is called *stretch*. It is defined as the length of the path in the tree, which is used to route an edge. Low-stretch spanning trees have been well-studied in the literature, and there are efficient algorithms that construct low-stretch spanning trees [58]. When the objective function is the sum of the congestion of the edges instead of the maximum congestion over the edges, then it becomes easy to translate it into the stretch setting and hence this problem has been studied well. But the maximum congestion problem, which we call the *spanning tree congestion* (STC) problem, has been less-studied.

The root of the STC problem dates back to at least 30 years ago [21, 144]. The STC problem was first formally defined by Ostrovskii [132] in 2004, and since then, a number of results about it have been published. STC is also interesting as a structural parameter of a graph. The study of STC of a graph class can reveal many structural properties of the graph class. Hence, STC has been actively studied recently in special classes of graphs, giving many bounds and complexity results for different graph classes [133, 105, 99, 98, 24]. For general graphs, it has been shown that the STC is at most $\mathcal{O}(n\sqrt{n})$ [119], and that this is tight [132], where n is the number of vertices.

A related topic to spanning tree congestion is the *connected partitioning*, that is, partitioning graphs into subsets which induce connected graphs. Usually, there are some additional size constraints on these subsets, specified during input. Finding a connected partition is often used as a subroutine to find low-congestion spanning trees. The idea is to partition a graph into balanced connected parts, and then recurse on each of these partitions.

Connected partitioning also has other natural applications in networks. Soltan, Yannakakis, and Zussman [145] describe the following application for power grids. In power grids, it is desirable to partition the network into small connected *islands* so that each island can operate independently for a while. This is called *power grid islanding* and is used to avoid cascading failures in power networks. It is also desirable that each of these islands are large enough. It is easy to see that this task maps to the connected partitioning of the graph representing the power grid network.

A classic result in connected partitioning is a theorem known as the Győri-Lovász theorem, proved independently by Győri [79] and Lovász [118] in the 1970s. Informally, the theorem says that it is possible to partition a k-connected graph into k many connected subgraphs, each of a specified size. The theorem was recently generalized to vertex-weighted graphs by Chen et al. [46]. We call this generalization as the generalized Győri-Lovász theorem. The generalized Győri-Lovász theorem is crucially used in some of our results for the spanning tree congestion.

1.4.1 Our Contribution

For STC of an n-vertex graph, there is a tight bound of $\Theta(n\sqrt{n})$ known [119, 132], as mentioned above. However, when the number of edges m is less than $n\sqrt{n}$, this upper bound is weaker than the trivial upper bound of m. We obtain refined bounds on the STC of general graphs in terms of both m and n. We show that STC of any graph is at most $\mathcal{O}(\sqrt{mn})$, which is a factor of $\Omega(\sqrt{m/n})$ better than the trivial bound of m. We present a polynomial-time algorithm that computes a spanning tree with maximum congestion $\mathcal{O}(\sqrt{mn} \cdot \log n)$. We also present an exponential-time algorithm for computing a spanning tree with maximum congestion $\mathcal{O}(\sqrt{mn})$; this algorithm runs in sub-exponential time when $m = \omega(n)$. We also show that this upper bound is tight by demonstrating graphs with STC at least $\Omega(\sqrt{mn})$, for almost all ranges of the average degree $2m/n$.

The STC of random graphs have been studied in the literature, especially by Ostrovskii [134]. Ostrovskii posed the question whether random graphs have an STC of $\Theta(n)$. We resolve this open problem here by providing upper and lower bounds on the STC of random graphs. Our bounds hold for a broader class of graphs than random graphs, more specifically, graphs which satisfy certain good expanding properties.

Our contribution in the area of connected partitioning is to give the first elementary and constructive proof for the generalized Győri-Lovász theorem by providing a *local search* algorithm with running time $\mathcal{O}^*(2^n)$. The proof given by Chen et al. [46] is non-constructive, and extends the advanced techniques from homology theory, which was used by Lovász in his proof of the classic version. Our proof on the other hand is an extension of the simple graph theoretic techniques used by Győri.

1.5 Structure of the Thesis

In Chapter 2, we provide some notations, definitions, conventions and brief descriptions of basic concepts used throughout the thesis. The Chapters 3, 4, 5, and 6 are the technical chapters of the thesis. Each of these chapters might have additional notations and preliminaries. Chapter 3 deals with biclique cover, biclique partition and related problems; Chapter 4 deals with the Hadwiger's conjecture; Chapter 5 deals with the rainbow coloring and its variants; Chapter 6 deals with the spanning tree congestion and the connected partition. Each chapter has its own introduction section, where we introduce the problems, give the background and related work, and summarize our results. Then we proceed to describe the results in detail in the subsequent sections of the chapter. At the end of each chapter, we dedicate a section for providing some of the related open problems.

1.6 Acknowledgement of Collaboration

Chapter 3 is based on the following paper:

L. S. Chandran, D. Issac, and A. Karrenbauer. On the Parameterized Complexity of Biclique Cover and Partition. In *11th International Symposium on Parameterized and Exact Computation (IPEC 2016)*, volume 63 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:13, Dagstuhl, Germany, 2017

Chapter 4 is based on the following paper:

L. S. Chandran, D. Issac, and S. Zhou. Hadwiger's Conjecture for Squares of 2-Trees. *European Journal of Combinatorics*, 76:159 – 174, 2019

Chapter 5 is based on the following papers:

L. S. Chandran, A. Das, D. Issac, and E. J. van Leeuwen. Algorithms and Bounds for Very Strong Rainbow Coloring. In *Latin American Symposium on Theoretical Informatics*, pages 625–639. Springer, 2018

P. Heggernes, D. Issac, J. Lauri, P. T. Lima, and E. J. van Leeuwen. Rainbow Vertex Coloring Bipartite Graphs and Chordal Graphs. In *43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018)*, volume 117 of *Leibniz*

International Proceedings in Informatics (LIPIcs), pages 83:1–83:13, Dagstuhl, Germany, 2018

L. S. Chandran, D. Issac, J. Lauri, and E. J. van Leeuwen. Rainbow Coloring and Forest number. Unpublished

Chapter 6 is based on the following paper:

L. S. Chandran, Y. K. Cheung, and D. Issac. Spanning Tree Congestion and Computation of Generalized Györi-Lovász Partition. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 32:1–32:14, 2018

CHAPTER 2

Notation and Preliminaries

In this chapter, we introduce some notations, definitions and concepts used throughout the thesis. In addition to this, each of the following chapters may introduce some additional notation and preliminaries.

2.1 Sets

We use $\mathbb{R}$ to denote the set of real numbers, $\mathbb{R}_{>0}$ to denote the set of positive real numbers, $\mathbb{Z}$ to denote the set of integers, $\mathbb{N}$ to denote the set of positive integers, $\mathbb{Z}_{\geq 0}$ to denote the set of non-negative integers and $\mathbb{Q}$ to denote the set of all rationals. We write $[n]$ for $\{1, 2, \ldots n\}$. For a singleton set $\{u\}$ we may sometimes write just u for convenience. For a set S, let 2^S denote the set of all subsets of S, let $\binom{S}{k}$ denote the set of all subsets of S of size k, and let $\binom{S}{\leq k}$ to denote the set of all subsets of S of size at most k. We use the operator $\sqcup$ to denote the disjoint union of sets.

2.2 Graphs

A graph G is defined as a pair of sets $(V(G), E(G))$ where $V(G)$ is called the set of vertices of G and $E(G)$, the set of edges of G is a multiset of pairs of vertices. A digraph G is defined as a pair of sets $V(G)$ and $E(G)$ where $V(G)$ is called the set of vertices of G and $E(G)$, the set of (directed) edges of G, is a multiset of *ordered* pairs of vertices. A graph (or digraph) G is called simple if the multiset $E(G)$ is a set. A graph (or digraph) is called a multigraph (multidigraph) if it is not simple. Whenever we say a graph, we will mean a simple graph unless explicity stated otherwise. For an edge $\{u, v\}$ in a graph we may use uv or vu to denote it, for convenience. We call u and v as the endpoints of edge uv. If there is an edge (directed edge in case of digraph) e between vertices u and v, we say that u is adjacent to v and that e is incident on both u and v. A directed edge $\overrightarrow{uv}$ is said to be an *incoming edge* of v and an *outgoing edge* of u. We use $\overrightarrow{uv}$ to denote a directed edge (u, v); we may omit the direction and just say uv or vu when the direction is not significant in the context or if the direction cannot be inferred from the context.

A graph G is called an *edge-weighted graph* or simply a *weighted graph*, if there is a weight function $w : E(G) \to \mathbb{R}$ on the edges. A graph G is called a *vertex-weighted graph*, if there is a weight function $w : V(G) \to \mathbb{R}$ on the vertices.

For a graph G and a set $S \subseteq V(G)$, we define its included neighborhood $N_G[S] := \{v \in V(G) : v \in S \textit{ or } \exists u \in S \textit{ such that } uv \in E(G)\}$. We also define the excluded neighborhood $N_G(S) := N_G[S] \setminus S$. The second neighborhoods are defined as: $N^2_G[S] := N[N[S]]$ and $N^2_G(S) := N^2_G[S] \setminus S$. We may choose to omit the subscript G when the graph is clear from the context. For vertex sets S_1 and S_2, we define $N_{S_2}(S_1) := N_G(S_1) \cap S_2$.

For any vertex v in a (di)graph G, the *degree* of v in G, denoted by $\mathsf{deg}_G(v)$, is defined as the number of vertices of G that are adjacent to v. If the graph is clear from the context, we may omit the subscript G. The *in-degree* and *out-degree* of a vertex v in a digraph G is defined as the number of incoming and outgoing edges respectively. For any graph G, $\Delta(G)$ denotes the maximum degree over all vertices in G. A vertex with degree 0, i.e, a vertex that do not have any edges incident on it is called an *isolated vertex.* A vertex with degree 1 is called a *pendant vertex.*

A graph H is called a subgraph of a graph G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. A graph H is called an induced subgraph of a graph G if $V(H) \subseteq V(G)$ and $E(H) = \{uv : u \in V(H), v \in V(H), uv \in E(G)\}$. For any $S \subseteq V(G)$, we define the subgraph induced by S on G, denoted by $G[S]$, as a graph H such that $V(H) = S$ and $E(H) = \{uv : u \in S, v \in S, uv \in E(G)\}$. For $S \subseteq V(G)$, we define $G \setminus S := G[V(G) \setminus S]$.

An *independent set* of a graph is a set of vertices that are pairwise non-adjacent to each other. A *coloring* of a graph G is a function $c : \mathbb{N} \to V(G)$. A *k-coloring* of a graph G is a coloring $c : [k] \to V(G)$. Given a coloring of the graph, the vertices having the same color are said to form a color class. A coloring is called a *proper coloring* if every pair of vertices having the same color are non-adjacent, i.e., every color class is an independent set. A proper coloring is called an *optimal coloring* if there is no proper coloring of the graph that uses fewer number of colors. The number of colors used by an optimal coloring is called the *chromatic number* of G, denoted by $\chi(G)$. A graph is said to be *k-colorable* if it admits a proper k-coloring. An *edge coloring* of a graph G is a function $c : \mathbb{N} \to E(G)$. A graph together with an edge coloring is called an *edge colored* graph. A *vertex cover* of a graph is a set of vertices S such that any edge has at least one endpoint from S. An *clique* of a graph is a set of vertices that are pairwise adjacent to each other. The clique number of a graph, denoted by $\omega(G)$ is the number of vertices in the largest clique of G. A graph is called a complete graph if the whole graph is a clique. We use K_t to denote a complete graph on t vertices. A vertex is called a *simplicial vertex* if its neighborhood induces a clique.

A graph H is called a *minor* of a graph G if a graph isomorphic to H can be obtained from a subgraph of G by contracting edges. An *H-minor* is a minor isomorphic to H, and a *clique minor* is a K_t-minor for some positive integer t, where K_t is the complete graph of order t. A graph is called *H-minor free* if it does not contain an H-minor.

A graph is called a *bipartite graph* if it is 2-colorable. The 2 independent sets corresponding to the 2 color classes of a bipartite graph are called its bipartitions. For a bipartite graph G, we denote the two vertex bipartitions by $L(G)$ and $R(G)$. For a subgraph H of bipartite graph G, $L(H)$ denotes the set of vertices of H that are in $L(G)$, and $R(H)$ denotes the set of vertices of H that are in $R(G)$. A complete bipartite graph or *biclique* is a bipartite graph G such that every pair of vertices u, v such that u is from $L(G)$ and v is from $R(G)$, is adjacent. We use $K_{a,b}$ to denote a complete bipartite graph G such that $|L(G)| = a$ and $|R(G)| = b$.

A *walk* of length k in a graph is an alternating sequence of vertices and edges, $v_0, e_0, v_1, e_1, v_2, \ldots, v_{k-1}, e_{k-1}, v_k$, which begins and ends with vertices such that each e_i is incident on v_i and v_{i+1}. We might omit the commas and just write $v_0e_0v_1e_1v_2 \ldots$ $\ldots v_{k-1}e_{k-1}v_k$. In a simple graph, the walk can be specified just by the vertices and we will just write $v_0v_1 \ldots v_k$. A *path* is a walk in which all vertices are distinct. A path on k vertices is called a k-path denoted by $\mathcal{P}_k$. The vertices v_1 and v_k are called the endpoints of the path and the other vertices are called the internal vertices of the path. For two paths $P_1 = v_1v_2 \dots v_k$ and $P_2 = w_1w_2 \dots w_r$, if $v_k = w_1$, we use P_1P_2 to denote the path $v_1v_2 \dots v_kw_2w_3 \dots w_r$. The length of a path is defined as the number of edges in it, unless otherewise metnioned. For a weighted graph, the length of the path is defined as the sum of weights of edges in the path, unless otherewise mentioned. For a pair of vertices, the shortest path between them is a path with smallest length possible. We use $\mathsf{dist}_G(u, v)$ to denote the length of shortest path between vertices u and v in graph G. We may omit the subscript G if the graph is clear from the context. The *diameter* of G, denoted by $\mathsf{diam}(G)$, is the maximum distance between any pair of vertices of G. A *cycle* is a walk in which the first and last vertices are same, and all other vertices are distinct. The length of a cycle is the number of vertices (equivalently, the number of edges) in C. A *k-cycle* means a cycle of length k. For a cycle C, we use $|C|$ to denote the length of C. An odd(even) path/cycle is a path/cycle on odd(even) number of vertices.

A graph is called *connected* if between any pair of vertices, there is a path. A *connected component* of a graph is a maximal subgraph that is connected. Any graph can be partitioned into a set of connected components. A *k-connected graph* is a graph that remains connected after the removal of any $k - 1$ or fewer of its vertices. A *2-connected component* of a graph is a maximal subgraph that is 2-connected. A *cut vertex* of G is a vertex whose removal increases the number of connected components of G. A *k-edge-connected graph* is a graph that remains connected after the removal of any k or fewer of its edges. A *bridge* of G is an edge whose removal increases the number of connected components of G.

A *forest* is a graph which does not have a cycle. A *tree* is a connected graph which does not have a cycle. A *spanning tree* T of a connected graph G is a tree T on vertices $V(G)$ such that $E(T) \subseteq E(G)$. A maximum induced forest of a graph G is an induced subgraph F of G that is a forest, such that the number of vertices in F is as large as possible. The *forest number* of a graph G, denoted by $f(G)$ is the number of vertices in any maximum induced forest of it.

Given a graph G, an edge set $E' \subseteq E(G)$ and 2 disjoint vertex subsets $V_1, V_2 \subset V(G)$, we let $E'(V_1, V_2) := \{v_1v_2 \in E' : v_1 \in V_1 \text{ and } v_2 \in V_2\}$.

2.2.1 Graph Classes

As we will be dealing with graph classes a lot, let us give a brief definition of some graph classes here. More definitions and properties will be added as needed when we handle these graphs. A detailed background on these graph classes can be found, for example, in the book by Brandstädt, Le, and Spinrad [28].

A graph is called a *chordal graph* if it contains no induced cycles of length at least 4. A chordal graph has a *simplicial ordering*, i.e., an ordering in which, for any vertex v, the neighbors of v that have a higher number than v form a clique. We call a graph G, a *k-simplicial graph*, if $V(G)$ has an ordering such that the higher numbered neighbors of each vertex can be partitioned into at most k cliques. A *split graph* is a graph whose vertices can be partitioned into one independent set and one clique. It is known that every split graph is a chordal graph.

A *domino graph* (see Figure 2.1) is the bipartite graph G with $L(G) = \{u_1, u_2, u_3\}$, $R(G) = \{v_1, v_2, v_3\}$ and $E(G) = \{u_1v_1, u_1v_2, u_2v_1, u_2v_2, u_2v_3, u_3v_2, u_3v_3\}$. A graph is

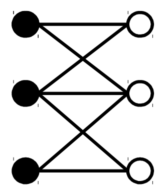


Figure 2.1: A domino graph

domino free if it does not contain the domino graph as an induced subgraph. A bipartite graph G is said to be a convex bipartite graph if $L(G)$ (or $R(G)$) can be enumerated such that for all $v \in L(G)$, the vertices adjacent to v are consecutive.

A graph is called *planar*, if it can be drawn on plane without any crossing edges. An equivalent characterization is that they are the class of graphs which do not contain K_5 or $K_{3,3}$ as a minor. The drawing of a planar graph on a plane is called a planar embedding of it. Any planar embedding of a graph divides the plane into regions called *faces.* A planar graph that has a planar embedding in which there is a face that contains all the vertices of the graph is called an *outerplanar graph.* A graph is an *apex graph* if it contains a vertex (called an *apex*) whose removal results in a planar graph.

For any graph G, the line graph of G is defined as the graph H where $V(H) := E(G)$ and $e_1e_2 \in E(H)$ if and only if the edges e_1 and e_2 are incident on a common vertex in G. For any graph G, the square graph of G, denoted by G^2, is defined as $V(G^2) = V(G)$ and for each $v_1, v_2 \in V(G)$, $v_1v_2 \in E(G^2)$ if and only if $\mathsf{dist}(v_1, v_2) \leq 2$. For any graph G, the p-th power of G, denoted by G^p, is defined as: $V(G^p) = V(G)$ and for each $v_1, v_2 \in V(G)$, $v_1v_2 \in E(G^p)$ if and only if $\mathsf{dist}(v_1, v_2) \leq p$. A graph G is called a bounded degree graph if $\Delta(G)$ is bounded by some constant.

Given a universe $\mathcal{U} = \{x_1, x_2, \ldots, x_n\}$ and a family $\mathcal{F} = \{S_1, S_2, \ldots, S_t\}$ of subsets of $\mathcal{U}$, the *intersection graph* $G(\mathcal{F})$ of $\mathcal{F}$ has vertex set $\{v_1, \ldots, v_t\}$, and there is an edge between two vertices v_i and v_j if and only if $S_i \cap S_j \neq \emptyset$. We call $\mathcal{F}$, a *representation* of $G(\mathcal{F})$. A disk graph is a graph that can be represented as the intersection of disks on an Euclidean plane. An *interval graph* is an intersection graph of intervals on the real line. A set of intervals representing an interval graph is called its *interval model.* The interval graph is *proper* if it has an interval model where no interval is properly contained in another. An equivalent definition of interval graphs is as follows: a graph is an interval graph if it is chordal and it contains no triple of non-adjacent vertices such that there is a path between every two of them that does not contain a neighbor of the third. Hence, interval graphs are subclasses of chordal graphs. A *unit interval graph* is an intersection graph of unit intervals on the real line. The class of unit interval graph and proper interval graphs turns out to be the same. A *circular arc graph* is an intersection graph of arcs of a circle. A chordal graph is an intersection graph of subtrees of a tree.

A *block* of a graph is a maximal 2-connected component. A *block graph* is a graph in which all blocks are cliques. Block graphs are also subclasses of chordal graphs. A *cactus graph* is a graph in which each block of the graph is a cycle or an edge; equivalently, every edge belongs to at most one cycle.

Let σ be a permutation of the integers between 1 and n. We define a graph G_σ on vertex set $[n]$ in the following way. Vertices i and j are adjacent in G_σ if and only if they appear in σ in the opposite order of their natural order. A graph on n vertices is a *permutation graph* if it is isomorphic to G_σ for some permutation σ of the integers between 1 and n. A graph is a *bipartite permutation* graph if it is both a bipartite graph and a permutation graph.

A *k-tree* is any graph that can be obtained starting from a K_k and then repeatedly applying the following operation any number of times: pick a K_k in the graph, extend it to a K_{k+1} by adding a new vertex and making it adjacent to all the vertices in the K_k. A *partial k-tree* is any graph that is a subgraph of a k-tree. The *treewidth* of G can be defined as the smallest k such that G is a partial k-tree. The class of chordal graphs are related to k-trees as follows: in the definition of k-trees, we only allow to pick same sized clique in every step, but if we allow to pick cliques of different sizes in different steps (and extend it to a clique of size one larger), then we get the class of chordal graphs.

An Erdős-Rényi random graph, denoted by $\mathcal{G}(n,p)$, is a graph on n vertices, and for each pair of vertices, an edge between them is included in the graph with probability p, where the probabilities of different pairs are independent from each other.

Three vertices of a graph form an asteroidal triple if every two of them are connected by a path avoiding the neighborhood of the third. A graph is *AT-free* if it does not contain any asteroidal triple.

2.3 Matrices

We refer to [149] for basic background about matrices and linear algebra. For an $m \times n$ matrix A, we say that the i^{th} row is $A_{i,:}$, the j^{th} column is $A_{:,j}$, and the entry corresponding to the i^{th} row and j^{th} column is A_{ij}. For $I_1 \subseteq [m]$ and $I_2 \subseteq [n]$, A_{I_1,I_2} denotes the submatrix of A where we pick the rows corresponding to the indices in I_1 and the columns corresponding to the indices in I_2.

The rank of a matrix over a field $\mathbb{F}$ is the number of linearly independent rows or columns in the matrix. A matrix $A \in \mathbb{F}^{m\times n}$ has rank r over $\mathbb{F}$ if and only if there exist $U \in \mathbb{F}^{m\times r}$ and $V \in \mathbb{F}^{r\times n}$ such that $UV = A$, where the arithmetic is over the field $\mathbb{F}$. The real rank of a matrix A is its rank over the real field $\mathbb{R}$.

The ℓ_p-norm of a matrix $A \in \mathbb{R}^{m\times n}$ is defined as $(\sum_{i\in[m],j\in[n]} A_{i,j}^p)^{1/p}$. We use $\odot$ operator to denote the matrix product of two binary matrices over the boolean semiring arithmetic.

The adjacency matrix of a graph G on n vertices is an $n \times n$ binary matrix A where $A_{ij} = 1$ if and only if there is an edge in G between the i^{th} vertex and the j^{th} vertex. The node-edge incidence matrix of a graph G on n vertices and m edges is an $n \times m$ binary matrix A where $A_{ij} = 1$ if and only if the i^{th} vertex is in the j^{th} edge.

2.4 Computational Problems

Here we define some problems commonly used throughout the thesis. Other problems will be defined as and when necessary.

The canonical NP-hard problem is the *boolean satisfiability* problem. For defining the problem, we need to first define some terms in boolean logic. A formula is in *conjunctive normal form* (CNF) or clausal normal form if it is a conjunction of one or more clauses, where a clause is a disjunction of literals; otherwise put, it is an AND of ORs. Such a formula is called a *CNF formula.* A boolean formula is called a *k-CNF formula* if each clause has at most k literals. An *assignment* to a boolean formula is an assignment of *True* or *False* values to the variables in the formula. An assignment is called a satisfying assignment if the formula after the assigning of the values evaluates to *True.* A formula is called satisfiable if it has at least one satisfying assignment. The SAT and 3SAT problems are defined as follows.

SAT
Input: A CNF formula ψ
Output: Is ψ satisfiable?

3SAT
Input: A 3-CNF formula ψ
Output: Is ψ satisfiable?

Another problem that appears throughout the thesis is the vertex coloring problem, whose decision and optimization versions are defined as follows.

k-Coloring
Input: A graph G
Output: Is G k-colorable?

Coloring
Input: A graph G
Output: The chromatic number of G

2.5 Complexity Theory

We assume familiarity with basics of complexity theory like the classes P and NP, the asymptotic notation, NP-completeness etc. We refer to [70, 12] for further background on the topic. We define here a couple of advanced complexity classes that appear in the thesis.

BPP (Bounded Probabilistic Polynomial Time) is the class of decision problems solvable by a probabilistic Turing machine in polynomial time with an error probability bounded away from 1/2 for all instances. BPTIME(f(x)) (Bounded Probabilistic Time) is the class of decision problems solvable by a probabilistic Turing machine in $f(x)$ time with an error probability bounded away from 1/2 for all instances, where x is the length of input encoding.

ZPP (Zero-error Probabilistic Polynomial Time) is the complexity class of problems for which a probabilistic Turing machine exists with these properties:

- It always returns the correct YES or NO answer.
- The running time is polynomial in expectation for every input.

We use the $\mathcal{O}^*$ notation to hide all polynomial factors in input size and $\tilde{\mathcal{O}}$ to hide logarithmic factors.

2.5.1 Parameterized Complexity

We refer to [51] for a detailed introduction to the topic. A *parameterized problem* is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a fixed, finite alphabet. An instance of the parameterized problem is given by a pair $(I, k) \in \Sigma^* \times \mathbb{N}$. Here k is called the parameter. An algorithm solves the problem if it can decide whether a given instance (I, k) is in L or not. A parameterized problem is said to be **fixed parameter tractable (FPT)**, if there exist an algorithm which can solve the problem in $\mathcal{O}^*(f(k))$ time, where $f : \mathbb{N} \to \mathbb{N}$ is any computable function. The class FPT is the collection of all parameterized problems that are fixed parameter tractable. The most common parameter of a decision problem is the size of the solution. For example, the problem k-VertexCover is to decide whether there is a vertex cover of size at most k for a given graph. It turns out that this problem is indeed fixed parameter tractable, i.e., the problem is in class FPT. But some problems are not fixed parameter tractable. For example the problem of whether a graph has a k-coloring is NP-complete for any $k \geq 3$ and hence cannot be in FPT. But NP-completeness may not be the only reason for a problem not being in FPT. For example, the problem k-Clique, which is to decide whether a given graph has a clique of size at least k, has no FPT algorithm so far and is believed not to have one. However the problem does have an algorithm that runs in time $n^{\mathcal{O}(k)}$. So the problem is polynomial time solvable for each fixed value of k but is unlikely to have an FPT algorithm. To capture such problems, a complexity hierarchy called the W-hierarchy was introduced, in particular the class $\mathsf{W[1]}$. The formal definition of these classes are through terms in circuit complexity. For the purposes of this thesis, we are mainly concerned with whether a problem is $\mathsf{W[1]}$-hard or not. A problem is $\mathsf{W[1]}$-hard if it has an FPT-reduction from k-Clique, where an FPT-reduction is defined as follows. A parameterized reduction from problem $L_1 \subseteq \Sigma_1^* \times \mathbb{N}$ to problem $L_2 \subseteq \Sigma_2^* \times \mathbb{N}$ is an algorithm that given an instance $(I_1, k_1) \in \Sigma_1^* \times \mathbb{N}$ produces an instance $(I_2, k_2) \in \Sigma_2^* \times \mathbb{N}$ such that $I_1 \in L_1$ if and only if $I_2 \in L_2$. The reduction is said to be an FPT-reduction if $k_2 \leq f(k_1)$ for some computable function $f : \mathbb{N} \to \mathbb{N}$ and the whole reduction runs in $g(k_1)h(|I_1|)$ time for some computable function $g : \mathbb{N} \to \mathbb{N}$ and some polynomial function $h : \mathbb{N} \to \mathbb{N}$. It is known that $\mathsf{FPT} \subseteq \mathsf{W[1]}$ but the converse is not known. It is widely believed that $\mathsf{FPT} \neq \mathsf{W[1]}$.

Conjecture 2.1. *$\mathsf{FPT} \neq \mathsf{W[1]}$.*

Hence, if a problem is $\mathsf{W[1]}$-hard, then it is unlikely to be in FPT. The above conjecture can be thought of as the equivalent of $\mathsf{P} \neq \mathsf{NP}$ in the parameterized world. Another related and stronger conjecture is the Exponential Time Hypothesis (ETH).

Conjecture 2.2 (Exponential Time Hypothesis). *There exist a constant $c > 0$ such that* 3SAT *on n variables cannot be solved in time $2^{o(cn)}$.*

Some of the lower bounds that we prove will be assuming the ETH.

Kernelization. Kernelization is a type of preprocessing algorithm. An algorithm A is called a kernelization algorithm or a kernel of a parameterized problem P if it takes as input an instance (I, k) of P and produces as output an instance (I', k') of the same problem such that $|I'| \leq f(k)$ and $k' \leq g(k)$ for some computable functions $f : \mathbb{N} \to \mathbb{N}$ and $g : \mathbb{N} \to \mathbb{N}$. We also call algorithm A an $f(k)$-kernel. When $f(k)$ is polynomial in k, we say that A is a polynomial kernel. It is easy to see that every parameterized problem that has a kernel is in class FPT. The converse also turns out to be true.

Treewidth. A very common parameter used in the area of parameterized algorithms is treewidth. It can be defined as the smallest k for which the graph is a partial k-tree. But the more useful definition is through tree decompositions. A *tree decomposition* of a graph G is a tree, T, with vertices $X_1, ..., X_n$, where each X_i is a subset of $V(G)$, satisfying the following properties:

- The union of all sets X_i equals V. That is, each vertex of the graph is contained in at least one vertex of the tree.
- If X_i and X_j both contain a vertex v, then all vertices X_k of T in the (unique) path between X_i and X_j contain v as well. Equivalently, the tree vertices containing vertex v form a connected subtree of T.
- For every edge vw in the graph, there is a subset X_i that contains both v and w.

The width of a tree decomposition is the size of its largest set X_i minus one. The treewidth $\mathsf{tw}(G)$ of G is the minimum width among all possible tree decompositions of G.

2.5.2 Approximation algorithms

We give a brief overview and refer to [157, 161] for a detailed introduction to the topic. An α-approximation algorithm is an algorithm for a maximization or minimization problem, that returns a solution that is worse than the optimum solution at most by a factor of α . The factor α is called the approximation ratio. For a maximization problem, it is defined as the maximum ratio of optimum solution to value of the solution returned by the algorithm over all possible input instances. For a minimization problem, it is defined as the maximum ratio of value of the solution returned by the algorithm to value of the optimum solution over all possible input instances. Observe that according to this convention, the approximation ratio is always greater than 1 and it is desirable to get as small approximation ratio as possible. We note that some articles in the literature defines the approximation ratio as the ratio of value of optimum to value of the solution of the algorithm for minimization problems in which case the ratio can be less than 1 and it is desirable to get as high approximation ratio as possible.

An approximation algorithm is called a polynomial time approximation algorithm if it runs in polynomial time. A PTAS (Polynomial Time Approximation Scheme) is an algorithm that for each fixed $\varepsilon > 0$ runs in time polynomial in the input size and has an approximation ratio of at most $(1 + \varepsilon)$. An FPTAS (Fully Polynomial Time Approximation Scheme) is an algorithm that for each $\varepsilon > 1$ runs in time polynomial in both the input size and $1/\varepsilon$, and has an approximation ratio of at most $(1 + \varepsilon)$. A problem is said to belong to the class APX if there is an α-approximation algorithm for it for some fixed constant α.

2.6 Other Notations and Conventions

We assume basic familiarity with fields, rings, semi-rings etc. We will use $\mathbb{R}$ to denote the real field. We denote the Galois Field of size 2 by $\mathbb{F}_2$. The boolean semi ring is defined as the set $\{0,1\}$ together with the logical and ($\wedge$), or ($\vee$) operators. Whenever we say log we mean $\log_2$ unless otherwise mentioned. For a function f from a set S to a value, and for any subset S' of S, we use $f(S')$ to denote $\Sigma_{x \in S} f(x)$, unless otherwise specified. When we say poly(x) we mean some function of x that is asymptotically smaller than some polynomial function in x. When we say polylog(x) we mean some function of x that is asymptotically smaller than $\log^c x$ for some constant c.

CHAPTER 3

Covering and Partitioning Edges with Bicliques

3.1 Introduction

A **biclique** (also known as complete bipartite subgraph) of a graph G is defined as any subgraph H of it such that H is bipartite, and every vertex in $L(H)$ is adjacent to every vertex in $R(H)$. We study the problems of covering and partitioning the edge set of graphs (especially bipartite graphs) with bicliques. The problems are formally defined as follows.

k-BicCover
Input: a graph G
Output: whether the edges of G can be covered by at most k bicliques

k-BicPart
Input: a graph G
Output: whether the edges of G can be partitioned into at most k bicliques

The corresponding optimization versions of the above decision problems, where we require to find the cover/partition having minimum number of bicliques, are called Bic Cover and BicPart respectively. In many of the applications, the input graph is bipartite. Hence we will focus mainly on the case when the input graph is bipartite. We will call the decision problems restricted to bipartite graphs as k-BipBicCover and k-BipBicPart respectively and the optimization problems as BipBicCover and BipBicPart respectively.

The biclique cover/partition problems have a long history. It was shown by Orlin in 1977 that k-BicCover is NP-complete [131], even in bipartite graphs. He also conjectured that k-BicPart is NP-complete, which was later answered in the affirmative in [87]. The minimum number of bicliques to cover the edges of a graph is also called the *bipartite dimension* [65], and Orlin called the minimum k that admits a partition of the edge set into k bicliques, the *bicontent* [131]. We prefer the terms *biclique cover number* and *biclique partition number*, respectively, to avoid any confusion.

The notion of biclique cover and partition can be extended to edge-weighted graphs (we call it biclique weighted cover/partition) in a natural way. We say that an edge having weight w is covered/partitioned by a set $\mathcal{B}$ of bicliques if it is present in *at least/exactly* w bicliques in $\mathcal{B}$. We call the edge-weighted versions of the problem as k-BicWtdPart and k-BicWtdCover respectively, and the respective bipartite versions as k-BipBicWtd Cover and k-BipBicWtdPart. The weighted problem can also be thought of as the generalization to multigraphs because we can think of the weights as multiplicities of the edges (but each biclique is allowed to cover only one multiplicity of each edge).

One of the main reason for us focusing on bipartite graphs is that the biclique cover/partition numbers of bipartite graphs are equivalent to the following important notions in matrix factorization:

Definition 3.1 (Boolean Rank)**.** *Given a matrix $A \in \{0,1\}^{m\times n}$, the boolean rank of A is the minimum k for which there exist $B \in \{0,1\}^{m\times k}$ and $C \in \{0,1\}^{k\times n}$ such that $A = B \odot C$, where $\odot$ denotes the matrix product over the boolean semiring. The pair (B, C) is called a rank-k boolean decomposition of A.*

Note that for boolean rank, the input matrix is constrained to be binary as the boolean product always result in binary values. But for the following notion of binary rank we allow the input matrix to be any integer matrix.

Definition 3.2 (Binary Rank)**.** *The binary rank of a matrix $A \in \mathbb{Z}^{m\times n}$ over a field $\mathbb{F}$ is defined as the minimum k for which there exist $B \in \{0,1\}^{m\times k}$ and $C \in \{0,1\}^{k\times n}$ such that $A = B \cdot C$ where $B \cdot C$ is the product using the arithmetic over the field $\mathbb{F}$. The pair (B, C) is called a rank-k binary decomposition of A over $\mathbb{F}$. When we say binary rank or binary decomposition without specifying the field, we mean the infinite real field $\mathbb{R}$.*

For translating the notions between bipartite graphs and matrices we define the following notions.

Definition 3.3 ((Weighted) bi-adjacency matrix, Equivalent (weighted) graph)**.** *For a bipartite graph G with $L(G) = \{u_1, u_2, \ldots, u_m\}$ and $R(G) = \{v_1, v_2, \ldots, v_n\}$, its bi-adjacency matrix is given by a matrix $A \in \{0,1\}^{m\times n}$ where $A_{ij} = 1$ if and only if $u_iv_j \in E(G)$. Also, G is called the equivalent graph of matrix A. If G has edge-weights $w : E(G) \to \mathbb{Z}_{\geq 1}$, then the weighted bi-adjacency matrix of G is given by a matrix $A \in \mathbb{Z}^{m\times n}$ where $A_{ij} = w(u_iv_j)$ if $u_iv_j \in E(G)$, and 0 otherwise. Also, G is called the equivalent weighted graph of matrix A.*

The following equivalence was proved by Gregory et. al. [77].

Fact 3.4. *[77] The biclique cover number of a bipartite graph G is equal to the boolean rank of its bi-adjacency matrix.*

The following equivalence is also easy to see and we give a proof in Section 3.2.3.

Lemma 3.5. *The biclique (weighted) partition number of a (weighted) bipartite graph G is equal to the binary rank of its (weighted) bi-adjacency matrix A. Moreover given a rank-k binary decomposition of A, a biclique (weighted) partition of G can be constructed in polynomial time.*

We define the parameterized boolean and binary rank problems as below.

k-BoolRank
Input: a matrix $A \in \{0,1\}^{m\times n}$
Output: whether boolean rank of A is at most k

k-BinRank($\mathbb{F}$)
Input: a matrix $A \in \mathbb{Z}^{m\times n}$,
Output: whether binary rank of A over field $\mathbb{F}$ is at most k

The problem k-BoolRank is equivalent to k-BipBicCover due to Lemma 3.4 and k-BinRank($\mathbb{R}$) is equivalent to k-BipBicWtdPart due to Lemma 3.5. The problem k-BipBicPart is equivalent to the special case of k-BinRank($\mathbb{R}$) when the input matrix A is binary. Also note that k-BinRank($\mathbb{F}$) is polynomial time solvable by gaussian elimination when $\mathbb{F} = \mathbb{F}_2$ and input matrix is binary.

We also briefly study the following approximate variant of binary rank problem for binary matrices.

(k, q)-BinRankApx($\mathbb{F}$)
Input: a matrix $A \in \mathbb{Z}^{m\times n}$, a field $\mathbb{F}$ and positive integers k and q
Output: whether there exist a matrix $A' \in \mathbb{Z}^{m\times n}$ such that $||A' - A||_0 \leq q$ and binary rank of A' over $\mathbb{F}$ is at most k

Closely related to k-BicCover and k-BicPart are the following problems about covering/partitioning edges of a (non-bipartite) graph with cliques. Some of our techniques can also be applied to these problems as shown later.

k-EdgeCliqCover
Input: a graph G
Output: whether the edges of G can be covered by at most k cliques

k-EdgeCliqPart
Input: a graph G
Output: whether the edges of G can be partitioned into at most k cliques

A set of cliques that cover the edges is called an **edge clique cover** (ECC) and a set of cliques that partition the edges is called an **edge clique partition** (ECP). The smallest number of cliques needed to cover the edges of G is called the *edge clique cover number* and the smallest number of cliques required to partition the edges is called *edge clique partition number* of G. We also extend the definitions to the edge weighted versions k-EdgeCliqWtdCover and k-EdgeCliqWtdPart, in a natural way as in the biclique problems.

3.1.1 Related Work

k-BicCover was shown to be NP-complete even for bipartite graphs by Orlin in 1977. He also conjectured the NP-completeness of k-BipBicPart. This conjecture was later affirmed by Jiang and Ravikumar [87]. They showed NP-completeness of a problem called *Normal Set Basis* which is equivalent to k-BipBicPart.

Parameterized Complexity. Gramm et al. [76] gave a 2^k-kernel for k-EdgeCliq Cover, thereby showing that the problem is FPT. Following the same approach, Fleischner et al. [66] showed that k-BicCover and k-BicPart are FPT, by giving kernels with at most 3^k vertices for general graphs and with at most 2^{k+1} vertices for bipartite graphs. They also gave an algorithm brute forcing over the kernalized instance, but they reported incorrect running times of $\mathcal{O}^*(2^{2k^2+3k})$. This mistake was observed by Nor et. al. [129] and they pointed out that the correct running time of this algorithm is $\mathcal{O}^*(2^{2^{2k}\log k+3k})$. For k-BicCover, the latter paper also improved the running time slightly to $\mathcal{O}^*(2^{k2^{k-1}+3k})$. They also showed improved running times of $\mathcal{O}^*(2^{2k^2+3k})$ for chordal bipartite graphs and $\mathcal{O}^*(2^{2k^2(\ell-1)+3k})$ for bipartite graphs that do not have a crown graph of order ℓ (see definition 3.36 for a definition of crown graphs) as an induced subgraph. Fleischner et al. [66] also studied the variants of covering/partitioning vertices with bicliques and showed that they are NP-complete for every $k \geq 3$, even for bipartite graphs. They also studied the more restricted version of biclique vertex cover/partition where at least one of the sides contain at most a fixed small number b of vertices, and showed that this problem is W[1]-hard. Mujuni and Rosamond showed [125] that k-EDGE CLIQPART has a k^2 kernel. Running a brute force algorithm on the kernelized instance gives a $\mathcal{O}^*(2^{k^3})$ algorithm for k-EDGECLIQPART. Cygan, Pilipczuk and Pilipczuk [52] showed that k-EDGECLIQCOVER has no $\mathcal{O}^*(2^{2^{o(k)}})$-time algorithm assuming ETH. They also showed that the same problem does not have a $2^{\delta k}$-kernel for some constant δ unless P =NP.

Approximation Algorithms. Simon in 1990 [143] showed that BICCOVER is NP-hard to approximate within a factor of 2. Gruber and Holzer [78] used the reduction in [143] to show that BICCOVER cannot be approximated within factors $n^{1/3-\varepsilon}$ and $m^{1/5-\varepsilon}$ in polynomial time unless P =NP. Later Chalermsook, Heydrich, Holm and Karrenbauer [32] improved the inapproximability factors to $n^{1-\varepsilon}$ and $m^{1/2-\varepsilon}$ [1]. Assuming a stronger hypothesis that $\mathsf{NP} \neq \mathsf{BPTIME}(2^{\text{polylog}\, n})$, they were able to show an even better inapproximability factor of $n/2^{\log^{\frac{7}{8}+\varepsilon} n}$. For BICPART, only APX-hardness is known, which follows due to a reduction from vertex cover given by Jiang and Ravikumar [87]. In [32], approximation factors of $\frac{n}{\sqrt{\log n}}$ were obtained with polynomial time algorithms for both problems. Bein et al. studied the approximability of the biclique vertex partition variant, where we want to partition the vertices instead of edges [19]. For this variant they showed that there is no constant approximation in polynomial time unless $P = NP$, and also gave a polynomial time 2-approximation for the case when the optimum is a small constant. EDGECLIQCOVER is inapproximable within a factor of $n^{1/2-\varepsilon}$ in polynomial time. This follows easily from a reduction from COLORING by Kou, Stockmeyer and Wong in 1978 [95].

Low Rank Matrix Approximation. Recently there has been much study in the approximate variant of boolean/binary matrix rank problems [68, 14, 67], where given a matrix $A \in \{0,1\}^{m\times n}$, we want to find matrices $B \in \{0,1\}^{m\times k}$ and $C \in \{0,1\}^{n\times k}$ such that the matrix $B \cdot C$ (product under the relevant arithmetic) is close to A. In other words, we want to approximate a given matrix A by a close enough matrix A' which has a low rank. Depending on the arithmetic used for product $B \cdot C$, we get different problems such as approximate binary rank, approximate boolean rank etc. The closeness of A' and A are usually measured in some ℓ_p-norm for some $p \geq 0$. Ban et al. [14] studied the *generalized binary ℓ_0-Rank-k approximation* problem where the objective is to minimize $||A' - A||_0$ and the product could be using any arithmetic specified during input. They gave a PTAS for the problem that runs in time $(\frac{1}{\varepsilon})^{2^{\mathcal{O}(k)}/\varepsilon^2} mn \log^{2^k} n$ where $(1+\varepsilon)$ is the approximation factor. They also showed that assuming ETH, the running time cannot be improved to $2^{1/\varepsilon^{o(1)}} \cdot 2^{n^{o(1)}}$ or $2^{2^{o(k)}} 2^{n^{o(1)}}$. The latter lower bound was proved using Corollary 3.12 from this work. Fomin, Golovach, Lokshtanov, Panolan and Saurabh [67]

[1]The paper also claimed the same result for BICPART but there was an error in this proof.

(independently from Ban et al.) gave an algorithm for the case of boolean rank which runs in $(\frac{1}{\varepsilon})^{2^{\mathcal{O}(k)}/\varepsilon^2} mn$. Fomin, Golovach and Panolan [68] gave a $2^{\mathcal{O}(k2^k \cdot \sqrt{q \log q})}(mn)^{\mathcal{O}(1)}$ algorithm for approximate boolean rank with absolute error q, i.e., the requirement is that $||A' - A||_0 \leq q$.

Special graph classes. BicCover and BicPart are known to be polynomial time solvable in domino-free bipartite graphs [7]. BicCover is known to be polynomial time solvable in convex bipartite graphs [120, 121, 69]. BicCover was shown to be NP-complete even for chordal bipartite graphs [126].

Other related problems. Two closely related problems are MaxBiclique and BalancedBiclique. In MaxBiclique, the objective is to find the biclique in a bipartite graph with maximum number of edges. The variant where you want to maximize the number of vertices is polynomial time solvable. In BalancedBiclique, we need to output a biclique that has equal number of vertices on both sides and the objective is to maximize the number of vertices. Recently, Manurangsi [122] proved that MaxBiclique and BalancedBiclique cannot have $n^{1-\varepsilon}$ approximation algorithms assuming the small set expansion hypothesis and that $\mathsf{NP} \neq \mathsf{BPP}$. Lin [116] showed that BalancedBiclique is $W[1]$-hard when parameterized by value of optimal solution. Contrastingly MaxBiclique is in FPT when parameterized by value of optimal solution due to a $\mathcal{O}^*(k^{\sqrt{k}})$ algorithm by Feng et al. [64].

3.1.2 Our contribution

We present an algorithm with better running times for k-BinRank($\mathbb{F}$) and hence for k-BipBicWtdPart, in particular for k-BipBicPart.

Theorem 3.6. *k-BinRank($\mathbb{F}$) has an algorithm which runs in $\mathcal{O}(2^{2k^2+2k} + mn(\log n + \log m))$ time.*

Corollary 3.7. *k-BipBicWtdPart and in particular k-BipBicPart has an algorithm which runs in $\mathcal{O}(2^{2k^2+2k} + |L(G)|\,|R(G)|\,(\log |L(G)| + \log |R(G)|))$ time.*

This drastically improves the previous best running time bound [66], which was $\mathcal{O}^*(2^{2^{2k}\log k + 3k})$ [129].[2]

Remark 3.8. *In k-BinRank($\mathbb{F}$), if the constraints on B and C are relaxed to $B \in S^{m\times k}$ and $C \in S^{k \times n}$ for some $S \subseteq \mathbb{F}$, then our algorithm in Theorem 3.6 will still work correctly but with a running time of $\mathcal{O}(|S|^{2k^2+2k} + mn(\log n + \log m))$.*

We also extend the algorithm to (k, q)-BinRankApx($\mathbb{F}$).

Theorem 3.9. *(k, q)-BinRankApx($\mathbb{F}$) has an algorithm that runs in $\mathcal{O}((2^{2kq} + 2^{2q\log q} + 2^{kq+q\log q})2^{2kq+2k^2+4k}(k + \log q) + mn(\log m + \log n))$ time.*

We use similar kind of techniques to get the following result for k-EdgeCliqPart.

[2]Note that the bound reported in [66] is inaccurate as also observed in [129].

Theorem 3.10. k-EdgeCliqPart *has an algorithm which runs in* $\mathcal{O}^*(2^{k^2+2k\log k+k})$ *time.*

Here the improvement is a bit less drastic, but still substantial. To the best of our knowledge, the previous best reported running time is $\mathcal{O}^*(2^{k^3})$ by Mujuni and Rosamond. But the k^2-kernel given by them (see Section 3.2.3), in fact implies a $\mathcal{O}^*(2^{\mathcal{O}(k^2\log k)})$ algorithm for the problem, as we point out it in Section 3.2.3. We remark that our algorithm for k-EdgeCliqPart does not extend to the k-EdgeCliqWtdPart problem.

The main technique that we use in the above algorithms might be interesting on its own. The technique is very simple and uses basic linear algebraic notions. The main idea is that we do not need to enumerate all the rows (or columns) of our target matrix decomposition, but only need to enumerate a set of few rows which can span all the other rows.

In contrast to the above positive results, we show that k-BicCover (and hence k-BicWtdCover and k-BoolRank) is much harder, by giving the following reduction, which is an adaptation of the reduction for k-EdgeCliqCover by Cygan et al. [52]

Theorem 3.11. *There exists a polynomial time reduction that, given a* 3*-SAT instance* ψ *on* n *variables and* m *clauses, produces a bipartite graph* G *with* $|L(G)| + |R(G)| = \mathcal{O}(n+m)$ *such that there exists a positive integer* $k = \mathcal{O}(\log n)$ *for which* G *has a biclique cover of size at most* k *if and only if* ψ *is satisfiable.*

This theorem immediately gives the following running time lower bounds for k-Bic Cover.

Corollary 3.12. k-BicCover *cannot be solved in time* $\mathcal{O}^*(2^{2^{o(k)}})$*-time unless the Exponential Time Hypothesis is false.*

This almost matches the best known upper bound for the running time of k-BicCover, which is $\mathcal{O}^*(2^{2^k\log k+2k+\log k}/k!)$ [129].
Another consequence of Theorem 3.11 is the following lower bound on size of the kernel for k-BicCover produced by any polynomial time algorithm.

Corollary 3.13. *There exists a constant* $\delta > 0$ *such that, unless* $P = NP$*, there is no polynomial time algorithm that produces a kernel for* k-BicCover *of size less than* $2^{\delta k}$.

We also give polynomial time approximation algorithms for BicCover and BicPart with improved approximation guarantees.

Theorem 3.14. BicCover *and* BicPart *have polynomial time approximation algorithms that gives an approximation factor of* $|V(G)|/\log_3|V(G)|$ *for an input graph* G*. For a bipartite graph* G*, the approximation factor can be improved to* $n/\log_2 n$ *where* $n = \min(|L(G)|, |R(G)|)$.

Finally we give an approximation algorithm for EdgeCliqPart. To the best of our knowledge, there is no polynomial time approximation algorithm known so far which achieve an approximation ratio of $|V(G)|^{2-\varepsilon}$ or $|E(G)|^{1-\varepsilon}$ where G is the input graph and ε is some positive constant.

Theorem 3.15. EdgeCliqPart *has a polynomial time approximation algorithm that gives an approximation factor of* $\min((\mathsf{ecp}(G)^2, |V(G)|^{4/3}, |E(G)|^{2/3})$ *for any graph* G.

3.2 Preliminaries

We use $\mathsf{bc}(G)$ and $\mathsf{bp}(G)$ to denote the biclique cover number and biclique partition number respectively of a graph G. It is clear that $\mathsf{bc}(G) \leq \mathsf{bp}(G)$ as any biclique partition is also a biclique cover. We use $\mathsf{bool}(A)$ to denote the boolean rank of A. and $\mathsf{bin}_{\mathbb{F}}(A)$ to denote the binary rank of A over $\mathbb{F}$. We might write bin for $\mathsf{bin}_{\mathbb{R}}$. We use ecc and ecp to denote the edge clique cover and partition numbers respectively.

3.2.1 Equivalence between biclique partition and binary rank

Proof of Lemma 3.5: Let $L(G) = \{u_1, u_2, \ldots, u_m\}$ and $R(G) = \{v_1, v_2, \ldots, v_m\}$. Consider a rank-$k$ binary decomposition (U, V) of A. We show that a biclique weighted partition of G of size k can be constructed as follows. We construct a set of bicliques $\mathcal{B} = \{B_1, B_2, \ldots, B_k\}$ as follows: $V(B_j) := \{u_i : U_{i,j} = 1\} \cup \{v_i : V_{j,i} = 1\}$. First we prove that B_ℓ indeed induces a biclique for each $\ell \in [k]$. Consider $u_i \in L(B_\ell)$ and $v_j \in R(B_\ell)$. Clearly then $U_{i,\ell} = 1 = V_{\ell,j}$. Then $A_{i,j} = U_{i,:}V_{:,j} \geq 1$. Hence, edge u_iv_j is present in G. Now, we prove that each edge is present in exactly as many bicliques as its weight. Consider an edge u_iv_j with weight w. This means that $A_{i,j} = w$. This implies $U_{i,:}V_{:,j} = w$. Since U and V are binary, this means that $|I := \{\ell \in [k] : U_{i,\ell} = 1 = V_{\ell,j}\}| = w$. But $u_i, v_j \in V(B_\ell)$ if and only if $\ell \in I$. Hence edge u_iv_j is covered exactly w times by $\mathcal{B}$.

Now, let us prove the other direction. Suppose we are given a biclique weighted partition $\mathcal{B} = \{B_1, B_2, \ldots, B_k\}$ of G. We show that a rank-k binary decomposition (U, V) of A can be constructed as follows: $U_{i,j} := 1$ if and only if $u_i \in V(B_j)$ and $V_{i,j} = 1$ if and only if $v_j \in V(B_i)$. Let $I = \{\ell \in [k] : U_{i,\ell} = 1 = V_{\ell,j}\}$. But $\ell \in I$ if and only if $u_i, v_j \in B_\ell$. Hence $|I|$ is exactly the number of bicliques which contain edge u_iv_j, which is equal to w as $\mathcal{B}$ is a biclique weighted partition of G. Then $U_{i,:}V_{j,:} = |I| = w$. $\square$

3.2.2 Review of the kernels for biclique cover and partition

Here, we briefly review the kernels for biclique cover and partition given by Fleischner, Mujuni, Paulusma, and Szeider [66]. We use this kernel as a subroutine in our FPT and approximation algorithms for biclique cover and partition. The kernelization is very simple and consists of a single reduction rule, which is the removal of twin vertices. For describing the reduction rule, we first define twin vertices.

Definition 3.16 (Twin vertices). *Two vertices v_1 and v_2 are said to be* twins *of each other if and only if $N(v_1) = N(v_2)$.*

The twin reduction rule is as follows.

Reduction rule 3.17 (Twin reduction). *If G contains twins v_1 and v_2, delete v_1 (or v_2).*

When we say we apply twin reduction to a graph G, we mean to apply Reduction rule 3.17 exhaustively, i.e., until there are no more twins in G. The twin reduction rule is very common technique in the area of kernelization. Fleischner et al. [66] used it to get kernels for biclique cover and partition. The following lemma shows that the reduction rule is sound with respect to k-BicCover and k-BicPart.

Lemma 3.18. *Let G be a graph, and G' be the graph obtained after applying twin reduction on G. Then $\mathsf{bp}(G) = \mathsf{bp}(G')$ and $\mathsf{bc}(G) = \mathsf{bc}(G')$. Also, given a biclique cover/partition of G' of size k, one can construct a biclique cover/partition of G of size k in time polynomial in the size of G.*

Proof. First we prove the statement for a single application of Reduction rule 3.17. If v_1 and v_2 are twins in G and $E(G \backslash v_1)$ can be covered/partitioned with k bicliques, then $E(G)$ can be covered/partitioned with k bicliques by adding v_2 to all the bicliques containing v_1. The converse is easy to see, i.e, if $E(G)$ can be covered/partitioned with k bicliques, then $E(G \setminus v_1)$ can be covered/partitioned with at most k bicliques. Thus $\mathsf{bp}(G) = \mathsf{bp}(G \setminus v_1)$ and $\mathsf{bc}(G) = \mathsf{bc}(G \setminus v_1)$. Since the twin reduction is a series of applications of Reduction rule 3.17, it follows that $\mathsf{bp}(G) = \mathsf{bp}(G')$ and $\mathsf{bc}(G) = \mathsf{bc}(G')$. Also, it is easy to see that constructing the cover/partition of G can be done in polynomial time once we have the cover/partition of G' by traversing each reduction step backwards. $\square$

The following lemma gives an upper bound on the size of the reduced instance.

Lemma 3.19 ([66])**.** *For a graph G that do not contain any twins, $|V(G)| \leq 3^{bc(G)} \leq 3^{bp(G)}$. For a bipartite graph G that do not contain any twins, $|L(G)|, |R(G)| \leq 2^{bc(G)} \leq 2^{bp(G)}$.*

The above lemma completes the 3^k-kernel for k-BicCover and k-BicPart because if $|V(G)| \geq 3^k$ for the graph G after applying twin-reduction, then we can immediately output that it is a NO instance.

Since we will be dealing with binary rank and boolean rank, which are the equivalents of biclique cover and partition, we restate the twin reduction in terms of matrices here.

Reduction rule 3.20 (Twin reduction in matrix terms). *If matrix A has two rows/columns that are identical, then remove one of them.*

This reduction rule is safe with respect to the k-BinRank($\mathbb{F}$) and k-BoolRank problems as shown by the following lemma.

Lemma 3.21. *Let A be an $m \times n$ integer matrix and A' be the matrix resulting after the application of Reduction rule 3.20 exhaustively on A. Then $\mathsf{bin}(A') = \mathsf{bin}(A)$ and $\mathsf{bool}(A') = \mathsf{bool}(A)$. (Note that in case of boolean rank assume that A is binary). The process of reducing A to A' takes $\mathcal{O}(mn(\log m + \log n))$ time and given a rank-k binary/boolean factorization of A', we can construct a rank-k binary/boolean factorization of A in $\mathcal{O}(mn)$ time.*

Proof. First we prove for one step in the reduction. Consider two identical rows in A. Let the identical rows be $A_{1,:}$ and $A_{2,:}$ without loss of generality. Let $B := A_{[2,m],:}$. Suppose (X, Y) is a rank-k binary/boolean decomposition of B. Note that X is a $(m-1) \times k$ matrix. Let X' be the $m \times n$ matrix defined as $X'_{1,:} := X_{1,:}, X'_{2,:} := X_{1,:}$ and for all $i \in [3, m]$, $X'_{i,:} := X_{i-1,:}$. It is easy to see that (X', Y) is a rank-k binary/boolean decomposition of A. Thus if B has a rank-k binary/boolean decomposition then so does A. The other direction is easy to see as follows. Suppose A has a rank-k binary/boolean factorization (X, Y). Then $(X_{[2,m],:}, Y)$ is a rank-k binary/boolean factorization of B. Thus if A has a rank-k binary/boolean decomposition then so does B. Hence $\mathsf{bin}(A) = \mathsf{bin}(B)$

and $\mathsf{bool}(A) = \mathsf{bool}(B)$. A similar argument can be shown for deletion of columns also. Since, the whole reduction is a series of such deletions, by transitivity it follows that $\mathsf{bin}(A) = \mathsf{bin}(A')$ and $\mathsf{bool}(A) = \mathsf{bool}(A')$.

The reduction procedure can be implemented in $\mathcal{O}(mn(\log n + \log m))$ time as follows: first sort all the rows in a lexicographic ordering; now the duplicate rows are in consecutive rows and one can remove them in linear time; now repeat the same for columns. Note that the step of deleting a column will not make two non-identical rows identical, because we keep a column that is identical to the deleted column undeleted, so if the rows had different entries in the deleted column, they will also have different entries in this undeleted column. Hence, we have to do the deletion procedure only once for rows and once for columns. It is clear that this procedure takes only $\mathcal{O}(mn \log n + nm \log m)$ time.

The process of constructing a factorization for A from a factorization of A' will take only $\mathcal{O}(mn)$ time as this only involves duplicating rows/columns. Note that we need to do some book-keeping during the reduction so that the reconstruction can be done efficiently, i.e., when we delete one of the two identical rows/columns, we should keep a note of which row/colum it was identical to. □

The equivalent of lemma 3.19 in matrix terms is as follows.

Lemma 3.22. *Consider a matrix $A \in \mathbb{Z}^{m \times n}$ that has all its rows distinct from each other and columns distinct from each other. Then $m, n \leq 2^{\mathsf{bin}(A)}$, and if A is a binary matrix then $m, n \leq 2^{\mathsf{bool}(A)}$.*

Proof. Let $k = \mathsf{bin}(A)$. Then there exist $B^* \in \{0,1\}^{m \times k}$ and $C^* \in \{0,1\}^{k \times n}$ such that $B^*C^* = A$. If 2 rows $B^*_{i,:}$ and $B^*_{j,:}$ of B^* are identical, then $A_{i,:} = B^*_{i,:}C^* = B^*_{j,:}C^* = A_{j,:}$, thus making 2 rows of A identical, which is a contradiction. Thus the rows of B^* are all distinct. But since rows of B^* are k-length binary vectors, there can only be 2^k distinct rows in B^*. Hence $m \leq 2^k$. Similarly it can be shown that $n \leq 2^k$. If A is binary, the result for boolean rank follows by replacing the product with boolean product in the above argument. □

We remark that for biclique partition, the kernelization can easily be extended to the edge-weighted case. In fact, we have shown that the kernelization works for binary rank of positive integer matrices, which corresponds to the weighted bipartite case. It is not very hard to see that the kernelization can be extended to the weighted general case as well, for biclique partition. On the other hand, we could not find a way to extend the kernelization to edge-weighted biclique cover. One of the reasons is that the boolean rank problem is only defined for binary matrices, and there is no equivalent matrix problem corresponding to the weighted bipartite biclique cover.

3.2.3 Review of the kernel for edge clique partition

Here we review the kernel having at most k^2 vertices given for k-EdgeCliqPart by Mujuni and Rosamond [125]. We use this kernel as a subroutine in our FPT and approximation algorithms for edge clique partition. We present the kernelization in a slightly different way to that in [125], but the core idea remaining the same. Additionally, we point out that the number of edges in the kernel is at most k^3, and that an $\mathcal{O}^*(2^{\mathcal{O}(k^2 \log k)})$ time algorithm for k-EdgeCliqPart is implied by the kernel.

The kernelization uses only one reduction rule, which is as follows. Recall that a simplicial vertex is a vertex whose neighborhood induces a clique.

Reduction rule 3.23. *If G has a simplicial vertex v, then remove v and the edges within $N[v]$ from G.*

The following lemma proves the soundness of the above rule.

Lemma 3.24. *Let v be a simplicial vertex in G. Let G' be the graph obtained by removing vertex v and the edges within $N[v]$ from G. Then G has an ECP of size k if and only if G' has an ECP of size $k-1$. Moreover, if $\mathcal{C}'$ is an ECP of G', then $\mathcal{C} = \mathcal{C}' \cup \{N[v]\}$ is an ECP of G.*

Proof. First we prove that if G has an ECP of size k then G' has an ECP of size at most $k-1$. Let $\mathcal{C}$ be an ECP of G. Let $\mathcal{C}_v$ be the set of all cliques in $\mathcal{C}$ containing v. Note that $|\mathcal{C}_v| \geq 1$, assuming that the trivial case where v is an isolated vertex does not occur. We show that $\mathcal{C} \setminus \mathcal{C}_v$ is an ECP of G'. Suppose it is not. Then there is an edge e in $E(G) \setminus E(N[v])$ that is not in any clique of $\mathcal{C} \setminus \mathcal{C}_v$. The edge e is not in any clique in $\mathcal{C}_v$ as any clique in $\mathcal{C}_v$ only has vertices from $N[v]$. But then e was not covered by $\mathcal{C}$ and hence $\mathcal{C}$ is not an ECP of G, which is a contradiction. Thus G' has an ECP of size at most $k-1$.

Next, we prove the other direction. Let $\mathcal{C}'$ be an ECP of G' and let $\mathcal{C} = \mathcal{C}' \cup \{N[v]\}$. We show that $\mathcal{C}$ is an ECP of G. Suppose it is not. Then there is an edge e of G that is not in any clique of $\mathcal{C}$. The edge e is not in $E(N[v])$ as then it would have been covered by the clique $N[v]$. But then e is an edge of G' that is not covered by $\mathcal{C}'$ and hence $\mathcal{C}'$ is not an ECP of G', which is a contradiction. Thus $\mathcal{C}$ is an ECP of G. ☐

We can repeat the Reduction rule 3.23 as long as there are simplicial vertices remaining. The following Lemma follows by the repeated application of Lemma 3.24.

Lemma 3.25. *Let G be a graph and G' be the resultant graph after applying the Reduction Rule 3.23 on G sequentially r times . Then $\mathsf{ecp}(G') = \mathsf{ecp}(G) - r$. Also, given an ECP of size k of G' we can construct an ECP of size $k+r$ of G in time linear in the size of G.*

Due to the above lemma, given an instance G of k-EdgeCliqPart, we can apply Reduction rule 3.23 safely on G until there are no simplicial vertices left. The following lemma proves a bound on the size of such a reduced instance.

Lemma 3.26. *Let G be a graph which does not contain any simplicial vertices. Then, $|V(G)| < (\mathsf{ecp}(G))^2$ and $|E(G)| < (\mathsf{ecp}(G))^3$.*

Proof. Let $\mathcal{C}$ be an optimal ECP of G. We first show that each clique in $\mathcal{C}$ has size at most $\mathsf{ecp}(G) - 1$. Suppose there was a clique $C \in \mathcal{C}$ such that $|C| \geq \mathsf{ecp}(G)$. Since there are no simplicial vertices in G, each vertex in C has at least 1 neighbor outside C. Thus each vertex in C has an edge incident on it which is not covered by C. For each vertex v in C, let e_v be such an edge. Let C_v be the clique in $\mathcal{C} \setminus C$ containing e_v. Observe that $C_u \neq C_v$ for any distinct u and v in $\mathcal{C}$ because, otherwise the edge uv will be covered twice in $\mathcal{C}$, once by C and second by $C_u = C_v$, which means $\mathcal{C}$ is not a partition of the edges. But since $|\mathcal{C} \setminus C| \leq \mathsf{ecp}(G) - 1$ and $|C| \geq \mathsf{ecp}(G)$, we arrive at a contradiction by a simple pigeon hole principle. Thus we conclude that size of each clique in $\mathcal{C}$ is at most $\mathsf{ecp}(G) - 1$. It follows immediately that $|V(G)| \leq \mathsf{ecp}(G)(\mathsf{ecp}(G) - 1)$. The number of edges in each clique in $\mathcal{C}$ is at most $(\mathsf{ecp}(G) - 1)(\mathsf{ecp}(G) - 2)/2$. Since each edge is present in at least one of the cliques in $\mathcal{C}$, we have that $|E(G)| \leq \mathsf{ecp}(G)(\mathsf{ecp}(G) - 1)(\mathsf{ecp}(G) - 2)/2$. $\square$

Due to the above lemma, we get a kernel for k-EdgeCliqPart with at most k^2 vertices and k^3 edges. This is because if after applying the Reduction rule 3.23 exhaustively on input graph G, it still has more than k^2 vertices or k^3 edges, then we can straightaway output that it is a NO instance.

Mujuni and Rosamond also reported an algorithm for k-EdgeCliqPart with running time $\mathcal{O}^*(2^{\Delta k})$ which implies a running time of $\mathcal{O}^*(2^{k^3})$ as $\Delta \leq k$ in the above kernel. But, their kernel in fact implies a better algorithm. For this, we need to use the additional property satisfied by the kernel that, each clique contains at most k vertices. Thus we can enumerate all possible solutions in $\binom{k^2+1}{k}^k = \mathcal{O}^*(2^{\mathcal{O}(k^2 \log k)})$ time.

We remark that we could not find a way to extend the above kernelization for edge clique partition to the weighted case.

3.3 Parameterized Algorithm for Binary Rank

This section is dedicated to the proof of Theorem 3.6 by giving an algorithm for k-BinRank($\mathbb{F}$) that runs in time $\mathcal{O}^*(2^{k^2+k})$. All the arithmetic operations, binary rank, and binary decomposition in this section are over the given field $\mathbb{F}$ unless otherwise mentioned.

Let A be the given $m \times n$ integer matrix. First we will give an algorithm with the assumption that, A has distinct rows and columns. Later, we will show how to reduce the case when there are duplicate rows or columns to this case. Because of Lemma 3.22, we can safely assume that $m, n \leq 2^k$ (otherwise we know A is a NO instance and we straightaway output that the binary rank of A is greater than k).

The pseudocode of the algorithm is given in Algorithm 1. If A is a YES instance, we know that there exist $B^* \in \{0,1\}^{m \times k}$ and $C^* \in \{0,1\}^{k \times n}$ such that $A = B^* C^*$. Note that the pair B^*, C^* may not be unique, but we fix some such pair. First, we guess a set of indices of k rows, $I = \{i_1, i_2, \ldots i_k\}$ of B^* that span the other rows of B^* (in line 1). That is, every other row of B can be expressed as a linear combination of rows $i_1, i_2, \ldots, i_k$ (over the field $\mathbb{F}$). Note that we do not need the rows $i_1, i_2, \ldots, i_k$ to be linearly independent, we just need them to span the other rows. Such k rows should exist because B^* has at most k columns and hence its rank in the real field is at most k. The number of possibilities for this guess is at most $\binom{m}{k} \leq \binom{2^k}{k} \leq 2^{k^2}$. Note that here we used that $m \leq 2^k$. Let $\tilde{B}$ and $\tilde{A}$ denote the sub-matrix of B^* and A respectively, limited to the rows $i_1, i_2, \ldots, i_k$. We also guess the entries of $\tilde{B}$ (also in line 1). Since $\tilde{B}$ is a binary matrix with k rows and k columns, the number of possibilities for this guess is at most 2^{k^2}.

Next, we find columns $c_1, c_2, \ldots, c_n$ of our target $C \in \{0,1\}^{k \times n}$ each independently (in line 5), such that they are compatible with $\tilde{B}$ and $\tilde{A}$, more specifically, we find a

Algorithm 1: Algorithm for the kernalized instance of k-BinRank($\mathbb{F}$)

```
Input      : A matrix $A \in \mathbb{Z}^{m\times n}$,
Assumption : $A$ has distinct rows and distinct columns
Output     : If binary rank of $A$ is at most $k$ then output $B \in \{0,1\}^{m\times k}$ and
             $C \in \{0,1\}^{k\times n}$ such that $BC = A$.
             Otherwise report that binary rank of $A$ is greater than $k$.
1  foreach $I = \{i_1, i_2, \cdots, i_k\} \subseteq \binom{[m]}{k}$, and $\tilde{B} \in \{0,1\}^{k\times k}$ do        // Loop 1
2      $\tilde{A} \leftarrow A_{I,:}$
3      for $j \in [n]$ do
4          By iterating over all $k$-length binary vectors,
5          find $c_j \in \{0,1\}^k$ such that $\tilde{B}c_j = \tilde{A}_{:,j}$ ;
6          if there is no such $c_j$, then go to the next iteration of Loop 1;
7      end
8      let $C$ be the matrix with $c_1, c_2, \cdots, c_n$ as the columns;
9      for $i \in [m]$ do
10         By iterating over all $k$-length binary vectors,
11         find $b_i \in \{0,1\}^k$ such that $b_i^T C = A_{i,:}$ ;
12         if there is no such $b_i$, then go to the next iteration of Loop 1;
13     end
14     let $B$ be the matrix with $b_1^T, b_2^T, \cdots, b_m^T$ as the rows;
15     output $B$ and $C$ and terminate.
16 end
17 report that binary rank of $A$ is greater than $k$ and terminate.
```

vector c_j such that $\tilde{B}c_j = \tilde{A}_{:,j}$ for each $j \in [n]$. We know that for a YES instance such a c_j should exist for all $j \in [n]$ as $C^*_{:,j}$ is one such c_j.

Next, we find rows $b_1^T, b_2^T, \ldots, b_n^T$ of our target $B \in \{0,1\}^{m\times k}$ each independently (in line 11). For a YES instance, such rows should exist as shown by the following lemma.

Lemma 3.27. *Let $C \in \{0,1\}^{k\times n}$ be such that $\tilde{B}C = \tilde{A}$. Then for each $i \in [m]$, there exists a vector $b_i \in \{0,1\}^k$ such that $b_i^T C = a_i^T$.*

Proof. We only need to show that at least one such b_i exist for each $i \in [m]$. We exhibit $B^*_{i,:}$ as the required b_i. We know

$$B^*_{i,:}C^* = A_{i,:} \tag{3.1}$$

and also that there exist $\lambda_1, \lambda_2, \ldots, \lambda_k \in \mathbb{R}$ such that

$$B^*_{i,:} = \sum_{t=1}^{k} \lambda_t B^*_{i_t,:} \tag{3.2}$$

Substituting this in (3.1) gives

$$\sum_{t=1}^{k} \lambda_t B^*_{i_t,:}C^* = A_{i,:} \tag{3.3}$$

But $B^*_{i_t,:}C^* = A_{i_t,:}$ and substituting this in (3.3) gives

$$\sum_{t=1}^{k} \lambda_t A_{i_t,:} = A_{i,:} \tag{3.4}$$

Now,

$$B^*_{i,:}C = \sum_{t=1}^{k} \lambda_t B^*_{i_t,:}C \tag{3.5}$$

But we know $B^*_{i_t,:}C = A_{i_t,:}$ for all $t \in [k]$ by the selection of $c_1, c_2, \dots, c_n$. By substituting this in (3.5), we have

$$\begin{aligned} B^*_{i,:}C &= \sum_{t=1}^{k} \lambda_t A_{i_t,:} && (3.6)\\ &= A_{i,:} && (3.7)\end{aligned}$$

where the latter equality follows from (3.4). □

Thus if A is a YES instance, then we have found a binary decomposition (B, C) of A of rank k. If we fail to find such a decomposition (which means that for each possibility of $i_1, i_2, \dots, i_k$ and $\tilde{B}$, we either failed to find a c_j for some $j \in [n]$ such that $\tilde{B}c_j = \tilde{A}$ or we failed to find a b_i for some $i \in [m]$ such that $b_i^T C = A_{i,:}$), then A has to be a NO instance and we output in line 17 that the binary rank of A is greater than k. Thus we have proved the correctness of Algorithm 1.

Next, we prove that Algorithm 1 runs in $\mathcal{O}(2^{2k^2+2k})$ time. First, let us estimate the number of iterations of Loop 1 in Algorithm 1. As discussed above, the number of possibilities for $i_1, i_2, \dots, i_k$ is at most 2^{k^2} and that for $\tilde{B}$ is also at most 2^{k^2}. Hence, the number of iterations of Loop 1 is at most $2^{k^2} \cdot 2^{k^2} = 2^{2k^2}$. The two inner loops only have at most n and m iterations respectively, each of which is at most 2^k. Each iteration of the inner loops takes $\mathcal{O}(2^k)$ time, as there are only 2^k k-length binary vectors. Hence, the total time taken by Algorithm 1 is in $\mathcal{O}(2^{2k^2+2k})$.

It remains to take care of the case when A has duplicate rows or columns. We remove the duplicate rows and columns by applying the Reduction rule 3.20 exhaustively on A. Let A' be the resultant matrix. We know all rows of A' are distinct and all columns of A' are distinct. We run Algorithm 1 on A'. If Algorithm 1 returns a rank-k binary factorization of A', then we compute a rank-k binary factorization of A as given in Lemma 3.21. Instead, if Algorithm 1 returns that the binary rank of A' is greater than k, then we output that the binary rank of A is greater than k. This is correct as $\mathsf{bin}(A) = \mathsf{bin}(A')$ by Lemma 3.21. The parts except the call to Algorithm 1 takes $\mathcal{O}(mn(\log m + \log n))$ time by Lemma 3.21. Hence the whole algorithm takes only $\mathcal{O}(2^{2k^2+2k} + mn(\log n + \log m))$ time. This completes the proof of Theorem 3.6.

3.4 Parameterized Algorithm for Approximate Binary Rank

In this section, we prove Theorem 3.9 by giving an algorithm for (k, q)-BinRankApx($\mathbb{F}$). We assume that all the arithmetic operations, binary rank etc. in this section are over the given field $\mathbb{F}$, although we will not explicitly mention it. First we show how to extend the kernelization rule that we described in Section 3.3 to the approximate version. The idea is similar, that we remove identical rows and columns, but we need to keep some extra book-keeping here. The reason is that one row of the reduced matrix may represent 2 or more entries of the original matrix, say p entries, and making an error in such an entry actually means making p errors. So we will keep track of the number of entries represented by an entry $A_{i,j}$, which we call the error cost $E_{i,j}$. Initially we can assume that all the error costs are 1. For two matrices A and A', we define $E(A, A')$ as $\sum_{i,j:A_{i,j} \neq A'_{i,j}} E_{i,j}$. We will solve the following more general problem:

(k, q)-BinRankApxCost($\mathbb{F}$)
Input: $A \in \mathbb{Z}^{m \times n}$ and an error cost matrix $E \in \mathbb{Z}^{m \times n}_{\geq 1}$
Output: whether there exist a matrix $A' \in \mathbb{Z}^{m \times n}$ such that $E(A, A') \leq q$ and binary rank of A' over $\mathbb{F}$ is at most k

The reduction rule for kernelization is as follows:

Reduction rule 3.28. *If the i^{th} and j^{th} rows of A are identical, delete j^{th} row from A and E and set $E_{i,\ell} := E_{i,\ell} + E_{j,\ell}$ for all $\ell \in [n]$. Similarly, if the i^{th} and j^{th} columns of A are identical, delete j^{th} column from A and E and set $E_{\ell,i} := E_{\ell,i} + E_{\ell,j}$ for all $\ell \in [m]$.*

For proving the correctness of the rule, first we need to show that it is fine to make only identical errors in the identical rows/columns.

Lemma 3.29. *Consider a matrix A whose i^{th} and j^{th} rows (or columns) are identical. If there is a matrix A' such that $E(A', A) \leq q$, then there is a matrix A'' such that $E(A'', A) \leq q$, the binary rank of A'' is at most the binary rank of A', and $A''_{i,:} = A''_{j,:}$ (or $A''_{:,i} = A''_{:,j}$ resp.).*

Proof. We will prove the lemma for rows and the proof for columns follows by symmetry. Assume without loss of generality that $E(A'_{i,:}, A_{i,:}) \leq E(A'_{j,:}, A_{j,:})$. Let A'' be the matrix obtained by replacing $A'_{j,:}$ with $A'_{i,:}$ in A'. It is easy to see that A'' has binary rank at most that of A' and that $E(A'', A) \leq E(A', A)$. ☐

Consider an instance (A, E) of (k, q)-BinRankApxCost($\mathbb{F}$). Suppose row i and row j are identical in A and we delete row j from A according to reduction rule 3.28. Let $\hat{A}$ be the matrix after deletion of row j and let $\hat{E}$ be the new error matrix modified as given in reduction rule 3.28. Suppose $\hat{A}'$ is a solution for instance $(\hat{A}, \hat{E})$. Let A' be the matrix obtained from $\hat{A}'$ by copying the i^{th} row and inserting it as the j^{th} row (increasing the indices of the following rows by 1). We prove A' has binary rank at most k as follows. Since $\hat{A}'$ has binary rank at most k, there exist $\hat{B} \in \{0, 1\}^{m \times k}$ and $\hat{C} \in \{0, 1\}^{k \times n}$ such that $\hat{B}\hat{C} = \hat{A}'$. Let B be the matrix obtained from $\hat{B}$ by copying the i^{th} row and inserting it as the j^{th} row (increasing the indices of the following rows by 1). It is clear that $B\hat{C} = A'$ and hence A' has binary rank at most k. By the construction of $\hat{E}$ and $\hat{A}$, it is easy to see that $E(A', A) = \hat{E}(\hat{A}', \hat{A}) \leq q$. Hence a solution of $(\hat{A}, \hat{E})$ can be converted to a solution of (A, E) in $\mathcal{O}(n)$ time. We also need to show that if $(\hat{A}, \hat{E})$ is a NO instance then so is (A, E). We prove the contrapositive. Suppose A' is a solution to (A, E). We can assume due to Lemma 3.29 that $A'_{i,:} = A'_{j,:}$. Let $\hat{A}' := A'_{[m]\setminus j,:}$. Then,

$$\begin{aligned}\hat{E}(\hat{A}', \hat{A}) &= E(A', A) - E(A'_{i,:}, A_{i,:}) - E(A'_{j,:}, A_{j,:}) + \hat{E}(\hat{A}'_{i,:}, \hat{A}_{i,:})\\ &= E(A', A) - \sum_{\ell : A_{i,\ell} \neq A'_{i,\ell}} E_{i,\ell} - \sum_{\ell : A_{j,\ell} \neq A'_{j,\ell}} E_{j,\ell} + \sum_{\ell : \hat{A}_{i,\ell} \neq \hat{A}'_{i,\ell}} \hat{E}_{i,\ell}\\ &= E(A', A) - \sum_{\ell : A_{i,\ell} \neq A'_{i,\ell}} E_{i,\ell} - \sum_{\ell : A_{i,\ell} \neq A'_{i,\ell}} E_{j,\ell} + \sum_{\ell : A_{i,\ell} \neq A'_{i,\ell}} E_{i,\ell} + E_{j,\ell}\\ &= E(A', A) \leq q\end{aligned}$$

Hence, $\hat{A}'$ is a solution to $(\hat{A}, \hat{E})$. Hence the reduction rule is correct. Note that we gave the argument for only deletion of rows, but it can be shown for columns also similarly. Also note that we need to do some more book-keeping, i.e. for each deleted row (or column), we need to remember the row (or column) that was identical to it, so that we can reconstruct the corresponding row (or column) in B. But this is very straightforward and the reconstruction procedure takes only $\mathcal{O}(mn)$ time. The whole kernelization can be implemented in $\mathcal{O}(mn(\log m + \log n))$ time, the same way as the one in Section 3.3.

After the reduction rule is exhaustively applied, we show that the number of rows and columns is at most $2^k + q$. The reason is that there are at most q rows or columns that can have errors, and hence if we remove those rows from A and A' the remaining part of the two matrices should be same. Hence there exist q rows (and columns) of A whose removal gives a matrix with binary rank at most k. Hence the remaining part can have at most 2^k rows by Lemma 3.19. Thus we have the following lemma.

Lemma 3.30. *If (A, E) is a YES instance of (k, q)-BinRankApxCost($\mathbb{F}$) and A has all rows and columns distinct then A has at most $2^k + q$ rows and at most $2^k + q$ columns.*

Now, we guess the error positions in A. Note that there can be only at most q error positions as each position has error cost at least 1. Since the total number of positions is at most $(2^k + q)(2^k + q)$, we have that the number of possible guesses is at most

$$\binom{(2^k + q)(2^k + q) + 1}{q} = \mathcal{O}(2^{2kq} + 2^{2q \log q} + 2^{kq + q \log q})$$

Once we have guessed the positions of errors, we can compute the required A' as follows: The positions where there are no errors, we know $A'_{i,j} = A_{i,j}$. It only remains to compute the entries of the positions where error occurs. Let (B, C) be a k-rank binary decomposition of A over $\mathbb{F}$. The entry $A'_{i,j}$ is completely determined by $B_{i,:}$ and $C_{:,j}$. Hence the entries at error positions of A' are completely determined by at most q rows of B and q columns of C. We guess these rows and columns which adds a multiplicative factor of at most 2^{2kq} to the running time. Now we have completely fixed A'.

Now it only remains to check whether A' has binary rank at most k. For this we can just call the algorithm for k-BinRank($\mathbb{F}$) on (A', k). This will run in time $\mathcal{O}(2^{2k^2+2k}(2^k + q)^2 \log(2^k + q))$ according to Theorem 3.6. Thus the total running time is

$$\mathcal{O}((2^{2kq} + 2^{2q \log q} + 2^{kq + q \log q})2^{2kq + 2k^2 + 4k}(k + \log q) + mn(\log m + \log n)).$$

This completes the proof of Theorem 3.9.

3.5 Parameterized Algorithm for Edge Clique Partition

The idea for the algorithm is similar to that of k−BicPart. We first reduce k-Edge CliqPart to a matrix problem and then solve the matrix problem. The algorithm used for solving the matrix problem is very similar to the one for k-BinRank($\mathbb{F}$). The core idea remains the same and there are only small technical differences.

For k−EdgeCliqPart, there is a kernel of size k^2 given by Mujuni and Rosamond [125], which we described in Section 3.2.3. Given an instance G of k-EdgeCliq Part, we apply the Reduction rule 3.23 exhaustively on G. Let G' be the resulting instance. Lemma 3.24 implies that G is an YES instance of k-EdgeCliqPart if and only if G' is an YES instance of $k' -$ EdgeCliqPart for some $k' \leq k$. Moreover, given a clique partition of size at most k' of G', in polynomial time one can construct a clique partition of size at most k of G. Lemma 3.26 implies that G' has at most k^2 vertices (otherwise we can straightaway output that G has edge clique partition number greater than k). Hence from now on we will assume that our input graph G is a reduced instance using the above kernelization, and hence that the number of vertices in G is at most k^2.

We remark that the twin reduction rule similar to biclique partition does not work here. To see this consider vertices a, b such that $N[a] = N[b]$. Suppose we delete b, and later put b into all the cliques in which a is contained. If there is an edge between a and b, then the edge ab will be possibly covered in many cliques, which is not what we want. If there is no edge between a and b then b cannot be in the same clique as a.

Now, we will translate the problem into a matrix problem. For this we need the following definitions.

Definition 3.31 (Equal Except in Diagonal, $=_{ED}$)**.** *Two $n \times n$ matrices A and B are said to be equal except in diagonal, denoted by $A =_{ED} B$ if and only if $a_{ij} = b_{ij}$ for all $i, j \in [n]$ such that $i \neq j$.*

Definition 3.32 (Binary Symmetric Decomposition (BSD))**.** *For a matrix $A \in \mathbb{Z}^{n\times n}$, a binary matrix $B \in \{0,1\}^{n\times k}$ is said to be a rank-k binary symmetric decomposition of A if $BB^T =_{ED} A$.*

The following lemma translates the edge clique partition problem into a matrix decomposition problem.

Lemma 3.33. *A graph G has an edge clique partition of size k if and only if its adjacency matrix A has a rank-k BSD. Moreover, given the rank-k BSD of A one can find an ECP of G in time polynomial in $|V(G)|$.*

Proof. Let $V(G) = \{v_1, v_2, \ldots, v_n\}$. Suppose G has an edge clique partition of size k. Let $C_1, C_2, \ldots, C_k$ be the edge clique partition. Let $b_1, b_2, \ldots b_k$ be vectors in $\{0,1\}^n$ defined as follows: for all $i \in [k]$ and $j \in [n]$, $(b_i)_j = 1$ if and only if clique C_i contains vertex v_j. Let B be the matrix whose rows are $b_1^T, b_2^T, \ldots, b_n^T$. For $i \neq j$, we have $b_i^T b_j = 1$, if and only if v_i and v_j are in the same clique C_ℓ for some $\ell \in [k]$, which happens if and only if there is an edge between v_i and v_j in G. Hence, $BB^T =_{ED} A$.

Now, let us prove the other direction. Suppose B is a rank-k BSD of A. Note that B is an $n \times k$ binary matrix. Define $C_1, C_2, \ldots, C_k \subseteq V(G)$ as $C_j := \{v_i : B_{i,j} = 1\}$. We first prove that each C_j induces a clique in G. Consider some pair of vertices

$v_i, v_\ell \in C_j$. We have $A_{i,\ell} = B_{i,:}(B^T)_{:,\ell} = B_{i,:}(B_{\ell,:})^T \geq 1$ where the last inequality is because $B_{i,j} = B_{\ell,j} = 1$. But since A is binary, we have $A_{i,\ell} = 1$. Thus any distinct pair of vertices v_i, v_ℓ in C_j have an edge between them. Thus each C_j induces a clique. It only remains to prove that every edge appears in some clique C_j. Consider any edge $v_i v_\ell \in E(G)$. Since $A_{i,\ell} = 1$, we have $B_{i,:}(B^T)_{:,\ell} = B_{i,:}(B_{\ell,:})^T = 1$. So there exist a $j \in [k]$ such that $B_{i,j} = B_{\ell,j} = 1$. But then for such a j, we have $v_i, v_\ell \in C_j$. Thus any edge $v_i v_\ell \in E(G)$ is in one of the cliques C_j for $j \in [k]$. □

Note that a trivial BSD for an adjacency matrix A is the corresponding node-edge incidence matrix. This corresponds to an edge clique cover of the graph obtained by taking each edge as a clique. The rank of this BSD is as large as the number of edges in the graph or the number of 1's in A. Nevertheless, this is an easy way to see that any symmetric binary matrix has a BSD of finite rank.

Now, we give an algorithm (Algorithm 2) that solves k-BSD in $\mathcal{O}^*(n^k 2^{k^2+k})$ time. We prove the correctness and running time in Lemma 3.34. The proof for Theorem 3.10 follows as : For solving the k-EdgeCliqPart problem, we first run the kernelization (which takes polynomial time), convert the kernalized instance to a k-BSD instance by Lemma 3.33, and then run Algorithm 2 on the k-BSD instance. Since for the kernalized instance $|V(G)| \leq k^2$, we have $n \leq k^2$ in the k-BSD instance and hence the total running time is in $\mathcal{O}^*(2^{2k\log k + k^2 + k})$.

Algorithm 2: Algorithm for finding rank-k BSD

```
Input   : A matrix $A \in \{0,1\}^{n\times n}$,
Output  : If $A$ has a rank-$k$ BSD then output a $B$ such that $BB^T =_{ED} A$,
          otherwise report that $A$ has no rank-$k$ BSD
1  foreach $I = \{i_1, i_2, \cdots, i_k\} \subseteq \binom{[n]}{k}$, $b_{i_1} \in \{0,1\}^k, b_{i_2} \in \{0,1\}^k, \ldots, b_{i_k}$ do
   // loop 1
2      let $\tilde{B}$ be the matrix whose rows are $b_{i_1}^T, b_{i_2}^T, \ldots, b_{i_k}^T$.
4      if $\tilde{B}\tilde{B}^T \neq_{ED} A_{I,I}$ then go to the next iteration of Loop 1
5      for $i \in [n] \setminus I$ in ascending order do                // loop 2
6          By iterating over all $k$-length binary vectors,
7          find $b_i \in \{0,1\}^k$ such that for all $j \in I \cup [i-1]$, $b_i^T b_j = A_{i,j}$ ;
8          if there is no such $b_i$, then go to the next iteration of Loop 1;
9      end
10     let $B$ be the matrix with $b_1^T, b_2^T, \cdots, b_n^T$ as the rows;
11     output $B$ and terminate.
12 end
13 report that $A$ has no rank-$k$ BSD and terminate.
```

Lemma 3.34. *Given a matrix $A \in \{0,1\}^{n\times n}$, Algorithm 2 finds the rank-k BSD of an input matrix $A \in n \times n$ if it exists in $\mathcal{O}^*(n^k 2^{k^2+k})$-time. If A does not have a rank-k BSD, then the algorithm reports so.*

Proof. The core idea is same as that of Algorithm 1. It is clear that Loop 1 has at most $n^k 2^{k^2}$ iterations and Loop 2 has at most n iterations each of which takes $\mathcal{O}(2^k)$ time. Hence the running time is clear.

Next we prove the correctness. Suppose A indeed has a rank-k BSD. We show that then the algorithm will successfully output a rank-k BSD. Since A has a rank-k BSD, there exist a $B^* \in \{0,1\}^{n\times k}$ such that $B^*B^{*T} = A' =_{ED} A$ for some $A' \in \{0,1\}^{n\times n}$. Note that there can be more than one such B^* but we fix one. B^* clearly has a rank (over standard real arithmetic) of at most k as it only has k columns. Therefore there exist k rows of B^* which can span all the other rows. Consider the iteration of Loop 1 for which the algorithm picks these k row indices as $\{i_1, i_2, \ldots, i_k\} = I$ and also picks $(B^*_{j,:})^T$ as b_j for all $j \in I$. The latter means that $\tilde{B} = B^*_{I,:}$. Clearly then $\tilde{B}\tilde{B}^T = B^*_{I,:}(B^*_{I,:})^T =_{ED} A_{I,I}$. Hence the **then** part of Line 4 is not executed and we enter Loop 2. We now prove that, in Line 7, the algorithm will successfully find a vector b_i for all $i \in [n] \setminus I$, with the specified condition (i.e., the condition that for all $j \in I \cup [i-1]$, $b_i^T b_j = A_{i,j}$). We do this by induction on i. Assume we were able to find such b_t for all $t \in [i-1] \setminus I$. Using this induction assumption, we show we can also find a b_i. We will exhibit $(B^*_{i,:})^T$ as the required b_i. We know that

$$B^*_{i,:}B^{*T} = A'_{i,:} \qquad \text{and} \tag{3.8}$$

$$B^*_{i,:} = \sum_{\ell\in I} \lambda_\ell B^*_{\ell,:} \tag{3.9}$$

where each $\lambda_\ell \in \mathbb{R}$. Substituting (3.9) in (3.8),

$$A'_{i,:} = B^*_{i,:}B^{*T} = \Sigma_{\ell\in I}\lambda_\ell B^*_{\ell,:}B^{*T} = \Sigma_{\ell\in I}\lambda_\ell A'_{\ell,:} \tag{3.10}$$

Taking $b_i = (B^*_{i,:})^T$, we have that for all $j \in I$,

$$b_i^T b_j = B^*_{i,:}(B^*_{j,:})^T \quad \text{(We are in the iteration where } b_j = (B^*_{j,:})^T \text{ for all } j \in I) \tag{3.11}$$

$$= A'_{i,j} = A_{i,j} \quad (A' \text{ and } A \text{ are different only in the diagonal}) \tag{3.12}$$

So it only remains to prove $b_i^T b_j = A_{i,j}$ for $j \in [i-1] \setminus I$. Consider any $j \in [i-1] \setminus I$.

$$\begin{aligned}
b_i^T b_j &= B^*_{i,:}b_j \\
&= \Sigma_{\ell\in I}\lambda_\ell B^*_{\ell,:}b_j && \text{(using (3.9))} \\
&= \Sigma_{\ell\in I}\lambda_\ell b_\ell^T b_j && \text{(We are in the iteration where } b_\ell = (B^*_{\ell,:})^T \text{ for all } \ell \in I) \\
&= \Sigma_{\ell\in I}\lambda_\ell A_{\ell,j} && \text{(due to induction assumption)} \\
&= \Sigma_{\ell\in I}\lambda_\ell A'_{\ell,j} && \text{(Since } j \notin I) \\
&= A'_{i,j} && \text{(Using 3.10)} \\
&= A_{i,j} && (A' \text{ and } A \text{ are different only in the diagonal})
\end{aligned}$$

Thus we have exhibited $B^*_{i,:}$ as the required b_i. Note that the actual b_i that the algorithm picks might be different than $B^*_{i,:}$; we only used this to prove the existence of at least one such b_i. Thus by induction, for all $i \in [n] \setminus I$, the algorithm finds a b_i with the required conditions. In Line 11, the algorithm outputs matrix B with these $b_1^T, b_2^T, \ldots b_n^T$ as the rows. Since we satisfied the condition in Line 7, we have that $b_i^T b_j = A_{ij}$ for all $i, j \in [n]$ and $i \neq j$. Hence, $BB^T =_{ED} A$.

Note that whenever we terminate via Line 11, we successfully produce a rank-k BSD of A. Hence if A does not have rank-k BSD we will terminate via Line 13 and hence correctly report that A does not have a rank-k BSD. □

3.6 Lower Bounds for Biclique Cover

In this section, we prove Theorem 3.11, by giving a reduction from 3SAT to k-BicCover with $k = \mathcal{O}(\log n)$ where n is the number of variables. The theorem has consequences for the complexity of k-BicCover as stated in Corollaries 3.12 and 3.13. We prove the corollaries first and then the theorem.

Proof of Corollary 3.12: If k-BicCover has a $\mathcal{O}^*(2^{2^{o(k)}})$-time algorithm, then by Theorem 3.11 we have $\mathcal{O}^*(2^{2^{o(\log n)}}) = \mathcal{O}^*(2^{o(n)})$-time algorithm for 3SAT, a contradiction to ETH. □

Proof of Corollary 3.13: We give a proof sketch and refer to [52] for the details where the authors prove a similar statement for k-EdgeCliqCover. By Theorem 3.11, we have an algorithm A that takes an instance of 3SAT and gives an equivalent instance of k-BicCover with parameter $k = \mathcal{O}(\log n)$. Suppose there is a kernelization algorithm B that produces a kernel with less than $2^{\delta k}$ size for some δ to be fixed later. Since k-Bic Cover is NP-complete, there exists an algorithm C that takes an instance of k-Bic Cover and gives an equivalent instance of 3SAT in polynomial time. By composing the algorithms A, B, and C and fixing the parameter δ appropriately, we get a polynomial time algorithm D that given a 3SAT instance as input, produces an equivalent smaller 3SAT instance as output. We can apply D repeatedly to solve 3SAT in polynomial time. Hence, algorithm B cannot exist. □

The proof of Theorem 3.11 gives a reduction from 3SAT to k-BicCover, which is a modification of the one given in [52] from 3SAT to k-EdgeCliqCover.[3] The main difference is that we introduce an additional gadget consisting of $\log_2 n$ domino graph gadgets (defined below) in order to make the reduction work for k-BicCover, where n is the number of variables in the input 3SAT formula.

Definition 3.35 (Domino Graph)**.** *A* **domino graph** *is the graph G with $L(G) = \{u_1, u_2, u_3\}$, $R(G) = \{v_1, v_2, v_3\}$ and $E(G) = \{u_1v_1, u_1v_2, u_2v_1, u_2v_2, u_2v_3, u_3v_2, u_3v_3\}$ (See Figure 2.1).*

The domino graph gadget replaces the independent set of size $\log_2 n$ used in [52], i.e., we replace each vertex there by a domino graph here. We also modify some of the adjacencies in the construction such that the graph becomes bipartite. Moreover, we have simplified the reduction of [52] by using a simple trick. The trick is to make one of the domino graphs special by adding edges between this domino graph and clause gadgets so that the biclique covering this domino graph corresponds to a satisfying assignment. For k-EdgeCliqCover, this corresponds to making one of the vertices in the independent set special by adding edges from it to the clause gadgets. We give the complete reduction for k-BicCover here for being self-contained.

In [52], the authors use cocktail party graphs as the main gadget in their reduction. We use the bipartite analogue called *crown graphs.* A crown graph is basically a complete bipartite graph minus a perfect matching. It is formally defined as follows.

[3]A parameter preserving reduction from k-EdgeCliqCover to k-BicCover would have been better than having to redo the whole reduction from scratch. We could not find any such reduction.

Definition 3.36 (Crown Graph, H_r)**.** *A crown graph on $2r$ vertices denoted by H_r is a bipartite graph with bipartitions $L(G) = \{u_1, u_2, \ldots, u_r\}$ and $R(G) = \{v_1, v_2, \ldots, v_r\}$ such that there is an edge from u_i to v_j iff $i \neq j$. In other words, the edges missing between $L(H_r)$ and $R(H_r)$ form a perfect matching given by $\{u_i v_i : i \in [r]\}$. (See $H = H_n$ in Figure 3.1.)*

If we pick exactly one vertex from each of the edges of the missing perfect matching of the crown graph, then we get a maximal biclique provided that we pick at least one vertex from each of the bipartitions. The complement of this vertex set also forms a maximal biclique. These pair of bicliques are called duplex bicliques, formally defined as follows.

Definition 3.37. *[Duplex Biclique[4]] A duplex biclique of a crown graph H_r is defined as a pair of bicliques $\{B_1, B_2\}$ such that $L(B_1) \cap L(B_2) = \emptyset, R(B_1) \cap R(B_2) = \emptyset$, and $L(B_1) \cup L(B_2) = L(H_r)$, and $R(B_1) \cup R(B_2) = R(H_r)$.*

We go on to define a duplex biclique cover as follows.

Definition 3.38 (Duplex Biclique Cover)**.** *A duplex biclique cover of a crown graph is defined as a set of duplex bicliques that together cover all the edges of the graph. When we say the size of a duplex biclique cover, we mean the number of bicliques in the cover, which is twice the number of duplex bicliques.*

We prove the following two lemmas about crown graphs.

Lemma 3.39. *H_r has a duplex biclique cover of size $2\lceil \log r \rceil$ that can be found in time polynomial in n.*

Proof. We exhibit such a biclique cover. Let $\ell = \lceil \log_2 r \rceil$. For any $x \in \left\{0, 1, \ldots, 2^\ell - 1\right\}$ and $j \in [\ell]$, let $\langle x \rangle_j$ denote the j-th bit of the ℓ-bit binary representation of x. For each $j \in [\ell]$, we define the j-th duplex biclique $\{T_j^1, T_j^2\}$ as follows: T_j^1 is the subgraph induced by $\{u_i : \langle i-1 \rangle_j = 1\} \cup \{v_i : \langle i-1 \rangle_j = 0\}$, and T_j^2 is the subgraph induced by $\{u_i : \langle i-1 \rangle_j = 0\} \cup \{v_i : \langle i-1 \rangle_j = 1\}$, where u_i and v_i are defined as in Definition 3.36. It is easy to see that $\{T_j^1, T_j^2\}$ is indeed a duplex biclique. It is also easy to see that any pair of vertices u_i, v_j such that $i \neq j$ should be present in at least one of the ℓ duplex bicliques. □

Lemma 3.40. *Given a duplex biclique $\{B_1, B_2\}$ of H_r such that $|L(B_1)| = |R(B_1)|$, we can in polynomial time find a duplex biclique cover $\mathcal{B}$ of H_r with size $2\lceil \log_2 r \rceil$ such that $\{B_1, B_2\}$ is one of the duplex bicliques in $\mathcal{B}$.*

Proof. We partition $[r]$ into 2 sets

$$J_1 = \{j \in [r] : u_j \in L(B_1)(\text{ and } v_j \in R(B_2))\} \qquad \text{and}$$
$$J_2 = \{j \in [r] : u_j \in L(B_2)(\text{ and } v_j \in R(B_1))\}.$$

Since $|L(B_1)| = |R(B_1)|$ and $|R(B_1)| = |L(B_2)|$ (the latter is because B_1 and B_2 are duplex bicliques), it follows that $|L(B_1)| = |L(B_2)|$. Hence r is even and moreover that

[4]Duplex Bicliques correspond to Twin Cliques in [52]. We use this name to avoid confusion with twin vertices.

$|J_1| = |J_2| = r/2$. Without loss of generality, we can reorder the indices of the vertices such that $J_1 = \{1, 2, \ldots, \frac{r}{2}\}$ and $J_2 = \{\frac{r}{2}+1, \frac{r}{2}+2, \ldots, r\}$. Let $\ell = \log_2 r$. We define the duplex biclique $\{T_i^1, T_i^2\}$ for all $i \in [\ell]$ the same way as in the proof of Lemma 3.39. It is clear that the duplex biclique $\{B_1, B_2\}$ is the same as the duplex biclique $\{T_1^1, T_1^2\}$. Thus, the set of bicliques $\{T_1^1, T_2^1, \ldots, T_\ell^1\} \cup \{T_1^2, T_2^2, \ldots, T_\ell^2\}$ gives the required duplex biclique cover. □

Let ψ be the input 3SAT formula with n variables and m clauses. Let $x_1, \ldots, x_n$ be the variables of ψ and $C_1, \ldots, C_m$ be the clauses. Let C_i^1, C_i^2, and C_i^3 denote the 3 literals of clause C_i. For $1 \leq a \leq 3$, we say that $C_i^a = (x_j, 1)$ if the a^{th} literal in clause C_i is the variable x_j appearing in positive form, and we say $C_i^a = (x_j, 0)$ if the a^{th} literal in C_i is the variable x_j appearing in negated form.

Assumptions about the input 3SAT **formula**: We assume that the number of variables n is a power of 2. We also assume that if the instance is satisfiable, then there is a satisfying assignment A such that half of the variables are assigned true in A and the other half false. These assumptions can be handled easily by introducing some extra variables as shown in [52]. Note that this increases the number of variables at most by a factor of 4.
Let ℓ be such that $2^\ell = n$. We have that $\ell \in \mathbb{Z}$ since n was assumed to be a power of 2. Before giving the reduction, we give the following useful definition.

Definition 3.41 (Bisimplicial Edge). *An edge uv is said to be bisimplicial with respect to a biclique B iff $N(u) \cup N(v) = L(B) \cup R(B)$.*

Now, we give the reduction from 3SAT to k-BicCover.
Construction: Given ψ, we construct a bipartite graph G. See Figure 3.1 for an illustration of the construction. A vertex with superscript u indicates that it belongs to $L(G)$, and a superscript v indicates that it belongs to $R(G)$. The edges of G are divided into two sets, a set of important edges E^{imp} and a set of free edges E^{free}. The number of bicliques required to cover E^{imp} will be different depending on whether ψ is satisfiable or not, whereas the number of bicliques required to cover E^{free} will depend only on the number of variables and clauses of ψ but not on whether ψ is satisfiable or not. There are 5 main gadgets in our construction of G as given below.

(1) A graph H isomorphic to the crown graph H_n: Let the vertices of $L(H)$ be $h_1^u, h_2^u, \ldots, h_n^u$ and that of $R(H)$ be $h_1^v, h_2^v, \ldots, h_n^v$. The edges of H are in E^{imp}. The vertices h_i^u and h_i^v correspond to the i-th variable of ψ. h_i^u corresponds to the variable in positive form and the vertex h_i^v corresponds to the variable in negative form.

(2) A set P of clause gadgets $P_1, \ldots, P_m$: Each P_i is an induced matching of size 3. Let $L(P_i) = \{p_{i1}^u, p_{i2}^u, p_{i3}^u\}$ and $R(P_i) = \{p_{i1}^v, p_{i2}^v, p_{i3}^v\}$ and let the 3 edges of P_i be $p_{i1}^u p_{i1}^v, p_{i2}^u p_{i2}^v$, and $p_{i3}^u p_{i3}^v$. These edges are in E^{imp}. For all $i \in [m]$, P_i corresponds to the clause C_i in ψ, and the 3 edges in P_i correspond to the 3 literals in the clause, i.e., edge $p_{ia}^u p_{ia}^v$ corresponds to literal C_i^a for $a \in \{1, 2, 3\}$.

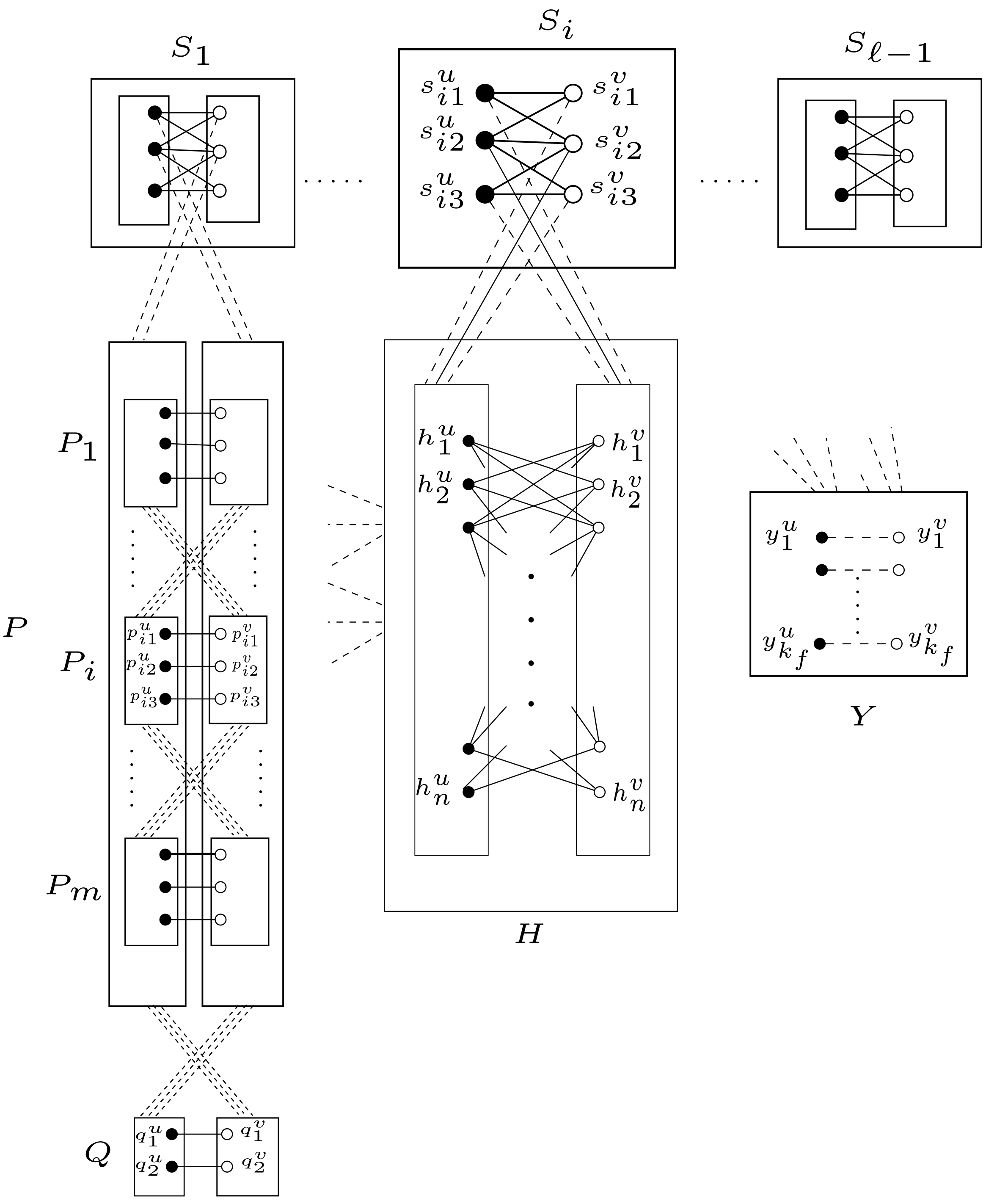


Figure 3.1: Illustration of the construction of G. The black nodes denote the vertices in $L(G)$, and the white nodes denote the vertices in $R(G)$. The solid edges represent edges in E^{imp}, and the dashed edges denote edges in E^{free}. The edges between P and H and those between Y and $G \setminus Y$ are not shown. The edges shown between S_i and H are present for all $i \in [\ell - 1]$, whereas the edges shown between S_1 and P are only present for S_1 and not for any S_i for $i \geq 2$.

(3) The *guard gadget* Q: Q is an induced matching of size 2. Let $L(Q) = \{q_1^u, q_2^u\}$ and $R(Q) = \{q_1^v, q_2^v\}$ and let the 2 edges of Q be $q_1^u q_1^v$, and $q_2^u q_2^v$. These edges are in E^{imp}.

(4) A set S of ℓ domino graphs $S_1, S_2, \ldots, S_\ell$ that are disconnected with each other: Let $L(S_i) = \{s_{i1}^u, s_{i2}^u, s_{i3}^u\}$ and $R(S_i) = \{s_{i1}^v, s_{i2}^v, s_{i3}^v\}$. The edges within each S_i are in E^{imp}.

(5) An induced matching Y of size k_f where $k_f = \mathcal{O}(\log n)$ will be fixed later (in Lemma 3.42): Y consists of edges $y_1^u y_1^v, \ldots, y_{k_f}^u y_{k_f}^v$, which are in E^{free}. For all $i \in [k_f]$, the edge $y_i^u y_i^v$ will be made bisimplicial with respect to a biclique $\tilde{B}_i^f$, which will be defined later. This is done to ensure that we need k_f bicliques to cover the free edges.

We also have the following edges between gadgets.

- Between H and S: For all $i \in [\ell]$ and $j \in [n]$, we add the following edges: $s_{i2}^u h_j^v$ and $s_{i2}^v h_j^u$ to E^{imp}; and, $s_{i1}^u h_j^v$, $s_{i3}^u h_j^v$, $s_{i1}^v h_j^u$, and $s_{i3}^v h_j^u$ to E^{free}.
- Between P_i and P_j: For all $i \neq j \in [m]$, add edges between all pairs of vertices u, v such that $u \in L(P_i)$ and $v \in R(P_j)$. These edges are in E^{free}.
- Between P and Q: For all $i \in [m]$, add edges between all pairs of vertices u, v such that $u \in L(Q)$ and $v \in R(P_i)$. Similarly, for all $i \in [m]$, add edges between all pairs of vertices u, v such that $u \in L(P_i)$ and $v \in R(Q)$. These edges are in E^{free}.
- Between H and P: For all $i \in [m], j \in [n]$ and $a \in [3]$, add edges between p_{ia}^u and h_j^v unless $C_i^a = (x_j, 1)$ and between p_{ia}^v and h_j^u unless $C_i^a = (x_j, 0)$. These edges are in E^{free}.
- Between S_1 and P: The only vertices in S that will have edges to any P_i are the 4 vertices $s_{11}^u, s_{12}^u, s_{11}^v$, and s_{12}^v. From s_{11}^u and s_{12}^u, add edges to all vertices in $R(P_i)$ for all $i \in [m]$. Similarly, from s_{11}^v and s_{12}^v, add edges to all vertices in $L(P_i)$ for all $i \in [m]$. These edges are in E^{free}.
- Between Y and $G \setminus Y$: These edges are added in such a way that edge $y_i^u y_i^v$ is bisimplicial w.r.t. a biclique that will be defined later. We will give the exact description of these edges after we define the bicliques B_i^f for $i \in [k_f]$. These edges belong to E^{free}.

Summary of $\mathbf{E}^{\mathsf{imp}}$: All the edges within H, S, and Q; all edges within each P_i; edges $s_{i2}^u h_j^v$ and $s_{i2}^v h_j^u$ for all $i \in [\ell - 1], j \in [n]$.
Summary of $\mathbf{E}^{\mathsf{free}}$: All the edges between P_i and P_j for $i \neq j$; all edges within Y; all edges between Y and $G \setminus Y$; edges $s_{i1}^u h_j^v$, $s_{i3}^u h_j^v$, $s_{i1}^v h_j^u$, and $s_{i3}^v h_j^u$ for all $i \in [\ell - 1], j \in [n]$; all edges between P and Q, between H and P, and between S_1 and P.

First, we show how to take care of the edges in E^{free} without interfering with the budget of E^{imp}. Let E^y be the set of all edges of G with at least one end point in $L(Y) \cup R(Y)$.

Lemma 3.42. *The edges in $E^{\mathsf{free}} \setminus E^y$ can be covered using $k_f = 4\log_2 n + 2\lceil \log_2 m \rceil + 6$ bicliques of G such that none of these bicliques contains an edge from E^{imp}, and these bicliques can be found in time polynomial in $n + m$.*

Proof. According to our construction, there are the following types of edges in $E^{\mathsf{free}} \setminus E^y$. For each of these types, we show how to cover it in polynomial time using bicliques that do not contain any edges from E^{imp} such that the total number of bicliques used is at most k_f.

- Edges between H and S: Such edges in E^{free} can be covered with 2 bicliques, B_1^{HS} and B_2^{HS} defined as follows. $L(B_1^{HS}) = L(H)$, $R(B_1^{HS}) = \{s_{i1}^v : 1 \le i \le \ell\} \cup \{s_{i3}^v : 1 \le i \le \ell\}$, $L(B_2^{HS}) = \{s_{i1}^u : 1 \le i \le \ell\} \cup \{s_{i3}^u : 1 \le i \le \ell\}$, and $R(B_2^{HS}) = R(H)$. From the construction of G, it is easy to see that both B_1^{HS} and B_2^{HS} are indeed bicliques, and none of them contains an edge in E^{imp}.

- Edges between P_i and P_j for $1 \le i < j \le m$: Such edges in can be covered with $2\lceil \log_2 m \rceil$ bicliques. Consider the subgraph of G given by the union of all such edges. Let this graph be G_1. The vertices p_{i1}^u, p_{i2}^u, and p_{i3}^u are twins (see Section 3.2.2 for definition) of each other in G_1 for all $i \in [m]$. Similarly, the vertices p_{i1}^v, p_{i2}^v, and p_{i3}^v are twins of each other in G_1 for all $i \in [m]$. Let G_2 be the graph obtained by applying twin-reduction (see Section 3.2.2 for definition) to G_1. It is clear that G_2 is isomorphic to the crown graph H_m. Hence, $E(G_2)$ can be covered by $\lceil 2\log_2 m \rceil$ bicliques in polynomial time by Lemma 3.39. Then, by using Fact 3.18, we can find $\lceil 2\log_2 m \rceil$ bicliques that cover $E(G_1)$. Since G_1 only contains edges from E^{free}, we do not cover any edges in E^{imp}.

- Edges between P and Q: We can cover these edges with 2 bicliques B_1^{PQ} and B_2^{PQ}, defined as $L(B_1^{PQ}) = L(P)$, $R(B_1^{PQ}) = R(Q)$; and, $L(B_2^{PQ}) = L(Q)$, $R(B_2^{PQ}) = R(P)$. Since there are no edges of E^{imp} between P and Q, we do not cover any edges in E^{imp}.

- Edges between H and P: We cover these edges with $4\log_2 n$ bicliques. We will show how to cover the edges between $L(H)$ and $R(P)$ with $2\log_2 n$ bicliques. Symmetrically, the edges between $R(H)$ and $L(P)$ can be covered with another $2\log_2 n$ bicliques. For $j \in [0, n-1]$, let $\langle j \rangle_i$ denote the i^{th} bit in the $\log_2 n$-bit binary representation of j. We will now give the description of a set of $2\log_2 n$ bicliques $\mathscr{B}^{HP} = \{B_1^{HP}, \ldots, B_{\log_2 n}^{HP}\} \cup \{\tilde{B}_1^{HP}, \ldots, \tilde{B}_{\log_2 n}^{HP}\}$ covering the edges between $L(H)$ and $R(P)$. We define $L(B_i^{HP}) = \{h_j^u : \langle j \rangle_i = 1\}$, $L(\tilde{B}_i^{HP}) = \left\{h_j^u : \langle j \rangle_i = 0\right\}$, $R(B_i^{HP}) = \bigcap_{u \in L(B_i^{HP})} N(u) \cap R(P)$, $R(\tilde{B}_i^{HP}) = \bigcap_{u \in L(\tilde{B}_i^{HP})} N(u) \cap R(P)$. It is clear that these are indeed bicliques from the definitions of $R(B_i^{HP})$ and $R(\tilde{B}_i^{HP})$. Since there are no edges of E^{imp} between H and P, we do not cover any edges in E^{imp}. We now show that we have covered every edge between $L(H)$ and $R(P)$. Suppose for the sake of contradiction that the edge $h_j^u p_{ia}^v$ was not covered. Let p_{ia}^v correspond to variable x_t. Recall that the only vertex in $L(H)$ that can possibly not have edge to p_{ia}^v is h_t^u. Edge $h_j^u p_{ia}^v$ not being covered by any biclique in $\mathscr{B}$ can happen only if every biclique in $\mathscr{B}^{HP}$ that contains h_j^u also contains h_t^u and if there is no edge between h_t^u and p_{ia}^v. But if every biclique in $\mathscr{B}$ containing h_j^u

also contains h^u_t, then $j = t$. This means that there is no edge between h^u_j and p^v_{ia}, which is a contradiction.

- Edges between S_1 and P: These edges can be covered with 2 bicliques B^{PS}_1 and B^{PS}_2, which is defined as follows. $L(B^{PS}_1) = L(P)$, $R(B^{PS}_1) = \{s^v_{11}, s^v_{12}\}$, $L(B^{PS}_2) = \{s^u_{11}, s^u_{12}\}$ and $R(B^{PS}_2) = R(P)$. Since there are no edges of E^{imp} between P and S_1, we do not cover any edges in E^{imp}.

□

We fix k_f as given by Lemma 3.42. By Lemma 3.42, we know that there are k_f bicliques that together cover all edges in $E^{\mathsf{free}} \setminus E^y$ and do not cover any edges in E^{imp}. We will call these bicliques $B^f_1, \ldots, B^f_{k_f}$.
Now, we give the description of the edges from Y to $G \setminus Y$. Recall that these edges are contained in $E^y \subset E^{\mathsf{free}}$. For each $i \in [k_f]$, we add edges from y^u_i to all the vertices in $R(B^f_i)$ and from y^v_i to all vertices in $L(B^f_i)$. Observe that now the edge $y^u_i y^v_i$ is bisimplicial with respect to B^f_i. This together with Lemma 3.42 gives the following lemma about the edge set E^{free}.

Lemma 3.43. *Let $k_f = 4\log_2 n + 2\lceil \log_2 m \rceil + 6$.*

(1) The edge set E^{free} can be covered using k_f bicliques of G.

(2) Any set of bicliques covering E^{free} has k_f bicliques that do not contain any edges from E^{imp}.

Proof. From Lemma 3.42, we know that $E^{\mathsf{free}} \setminus E^y$ can be covered by k_f bicliques that do not cover any edges from E^{imp}. Given these k_f bicliques $B^f_1, \ldots, B^f_{k_f}$, we extend them to the bicliques $\tilde{B}^f_i, \ldots, \tilde{B}^f_{k_f}$ as follows to cover all the edges of E^{free} : $L(\tilde{B}^f_i) = L(B^f_i) \cup \{y^u_i\}$, and $R(\tilde{B}^f_i) = R(B^f_i) \cup \{y^v_i\}$. It is clear that $\tilde{B}^f_1, \ldots, \tilde{B}^f_n$ are all indeed bicliques, and that they together cover all edges of E^{free}.

Since the edges within Y form an induced matching of size k_f, no two of them can be present in the same biclique. Hence, we need at least k_f bicliques to cover the edges in E^{free}. We now show that if a biclique contains an edge from E^{imp}, it cannot have an edge from $E[Y]$, which will complete the proof of the lemma. Suppose the edge $y^u_i y^v_i$ and an important edge $z^u z^v \in E^{\mathsf{imp}}$ are in the same biclique for the sake of contradiction. But since $N(y^u_i) = R(B^f_i) \cup \{y^v_i\}$, we have that $z^v \in R(B^f_i)$. Symmetrically, we can argue that $z^u \in L(B^f_i)$. Then, however, B^f_i contains the important edge $z^u z^v$, which is a contradiction. □

We set our budget k as $k_f + 2\ell + 2$, which means a budget of $2\ell + 2$ for the edges in E^{imp} due to Lemma 3.43. Now, we argue the completeness of the reduction in the following Lemma.

Lemma 3.44. *If ψ is satisfiable, then the edges in E^{imp} can be covered by $2\ell + 2$ bicliques of G. (These bicliques might contain some edges from E^{free} as well).*

Proof. We know that there exists a satisfying assignment A of ψ such that exactly half of the variables are assigned *true* in A. Each clause C_i has at least one literal which satisfies the clause. For each clause C_i, we fix one such literal. This literal corresponds to one of the 3 edges of P_i. Let us denote this edge by e_i.

We use two bicliques, B_1^g and B_2^g, to cover the 2 guard edges of Q and 2 edges from each P_i. Each of B_1^g and B_2^g covers 1 edge from Q and 1 edge from each P_i. It is clear that this can be done. Now, each P_i has one edge still to be covered. We will assign B_1^g and B_2^g such that the edge left uncovered in P_i is e_i, i.e., the literal corresponding to this edge evaluates to *true* in A.

Let B_1 be the biclique defined as follows: $L(B_1) = \{h_i^u : A(x_i) = true, i \in [n]\}$ and $R(B_1) = \{h_i^v : A(x_i) = false, i \in [n]\}$. Also, define $\bar{B}_1$ as the biclique defined by the vertex sets $L(\bar{B}_1) = L(H) \setminus L(B_1)$ and $R(\bar{B}_1) = R(H) \setminus R(B_1)$. B_1 and $\bar{B}_1$ are indeed bicliques of H because no literal evaluates to both *true* and *false*, and thus, the missing edges corresponding to the missing perfect matching in H are avoided. Likewise, they are duplex bicliques due to the manner in which $\bar{B}_1$ is defined. Moreover, $|L(B_1)| = |R(B_1)|$ since A has half of the variables assigned $true$ and the other half $false$. Therefore, by Lemma 3.40, there exist $\ell - 1$ other duplex bicliques $\left\{B_2, \bar{B}_2\right\}, \ldots, \left\{B_\ell, \bar{B}_\ell\right\}$ such that $B_1, \bar{B}_1, B_2, \bar{B}_2, \ldots, B_\ell$, and $\bar{B}_\ell$ together cover $E(H)$. Now we extend these bicliques with additional vertices so that these bicliques together with B_1^g and B_2^g cover E^{imp}, which is done as follows. For $2 \le j \le \ell$, we define biclique B'_j as $L(B'_j) = L(B_j) \cup \left\{s_{j1}^u, s_{j2}^u\right\}$ and $R(B'_j) = R(B_j) \cup \left\{s_{j1}^v, s_{j2}^v\right\}$. For $1 \le j \le \ell$, we define $\bar{B}'_j$ as $L(\bar{B}'_j) = L(\bar{B}_j) \cup \left\{s_{j2}^u, s_{j3}^u\right\}$ and $R(\bar{B}'_j) = R(B_j) \cup \left\{s_{j2}^v, s_{j3}^v\right\}$. B'_1 is defined as $L(B'_1) = L(B_1) \cup \{s_{11}^u, s_{12}^u\} \cup \bigcup_{i \in [m]} L(e_i)$ and $R(B'_1) = R(B_1) \cup \{s_{11}^v, s_{12}^v\} \cup \bigcup_{i \in [m]} R(e_i)$. It is clear that each B'_i and $\bar{B}'_i$ is indeed a biclique of G and that the bicliques $B'_1, B'_2, \ldots, B'_\ell, \bar{B}'_1, \bar{B}'_2, \ldots, \bar{B}'_\ell, B_1^g$, and B_2^g together cover E^{imp}. Hence, E^{imp} can be covered by $2\ell + 2$ bicliques of G. ☐

Now, we argue the soundness of our reduction in the next Lemma.

Lemma 3.45. *If E^{imp} can be covered by using $2\ell+2$ bicliques of G, then ψ is satisfiable.*

Proof. The edge set $M = \{q_1^u q_1^v, q_2^u q_2^v\} \cup \left\{s_{j1}^u s_{j1}^v : j \in [\ell]\right\} \cup \left\{s_{j3}^u s_{j3}^v : j \in [\ell]\right\}$ forms an induced matching of size $2\ell+2$ in G. Recall that all these edges are in E^{imp}. Since no two edges of an induced matching can be contained in the same biclique, each edge in M has to be covered by a distinct biclique. Let the biclique that covers $q_1^u q_1^v$ be B_1^g and the one that covers $q_2^u q_2^v$ be B_2^g. Let B_j and $\bar{B}_j$ be the bicliques covering the edges $s_{j1}^u s_{j1}^v$ and $s_{j3}^u s_{j3}^v$ respectively. Since we have already used our budget of $2\ell+2$, all the edges in E^{imp} must be covered by at least one biclique in $\mathscr{B} = \left\{B_g^1, B_g^2\right\} \cup \{B_1, B_2, \cdots B_\ell\} \cup \left\{\bar{B}_1, \bar{B}_2, \cdots, \bar{B}_\ell\right\}$. The only possible bicliques in $\mathscr{B}$ that can contain s_{j2}^u or s_{j2}^v are B_j and $\bar{B}_j$. That means, edges between s_{j2}^u and H and edges between s_{j2}^v and H have to be covered by $\{B_j, \bar{B}_j\}$. Moreover, they have to be partitioned by $\left\{B_j, \bar{B}_j\right\}$ for the following reason: if the edge $s_{j2}^u h_i^v$ appears in both bicliques B_j and $\bar{B}_j$, then the edge $s_{j2}^v h_i^u$ cannot appear in any of the two bicliques as there is no edge between h_i^u and h_i^v; and symmetrically, if the edge $s_{j2}^v h_i^u$ appears in both bicliques B_j and $\bar{B}_j$, then the edge $s_{j2}^u h_i^v$ cannot appear in any of the two bicliques. Combined with the fact that $N(\left\{s_{j2}^u, s_{j2}^v\right\}) = L(H) \cup R(H)$, we get that $\{B'_j, \bar{B}'_j\}$ is a duplex biclique, where B'_j and $\bar{B}'_j$ are the intersection of the bicliques B_j and $\bar{B}_j$, respectively, with H.

B^g_1 and B^g_2 can each cover at most 1 edge from each P_i. Hence, there is at least 1 edge of each P_i that must be covered by $\mathscr{B} \setminus \left\{B^1_g, B^2_g\right\}$. Let us fix one such edge for each P_i and call it e_i. Since, there are no edges from end points of e_i to any S_j for $j \geq 2$, we know that each e_i must be covered by B_1 or $\bar{B}_1$. But since end points of e_i are not adjacent to s^u_{13} and s^v_{13}, e_i cannot be covered by $\bar{B}_1$. Thus, each e_i has to be covered by B_1.

Now, we construct an assignment A according to B_1 as follows. For each $i \in [n]$, since $\{B_1, \bar{B}_1\}$ is a duplex biclique, B_1 contains exactly one among h^u_i and h^v_i. If $h^u_i \in L(B_1)$, then we assign $x_i = true$ in the assignment A. Otherwise, i.e, if $h^v_i \in R(B_1)$, then we assign $x_i = false$ in A. We claim that A must be a satisfying assignment for ψ, which can be observed as follows. Consider an arbitrary clause C_j. Let x_i be the variable corresponding to e_j. Suppose x_i occurs in positive form in C_j. From the construction of edges between H and P, we know that there cannot be an edge from h^v_i and end points of e_j. Hence, B_1 cannot contain h^v_i. But, since B_1 should contain one of h^u_i and h^v_i, it should contain h^u_i. This means that we assigned $x_i = true$ in A and hence A satisfies clause C_j. Symmetrically, we can argue that if x_i occurred in the negative form in C_j, then we would have assigned $x_i = false$ in A. Thus, A satisfies all the clauses of ψ. □

Proof of Theorem 3.11: We know that $E(G) = E^{\mathsf{free}} \cup E^{\mathsf{imp}}$. First we show that if ψ is satisfiable then we can cover $E(G)$ with $k = k_f + 2\ell + 2$ bicliques. So, assume ψ is satisfiable. Then, by Lemma 3.43(1), there is a set of k_f bicliques that covers E^{free}. By Lemma 3.44, there is a set of $2\ell + 2$ bicliques that cover E^{imp}. Thus, $E(G)$ can be covered with k bicliques.

Now, we show that if we can cover $E(G)$ with k bicliques then ψ is satisfiable. So, assume we can cover $E(G)$ with k bicliques. From Lemma 3.43(2), we know that we need at least k_f bicliques to cover E^{free} and that at least k_f many of the bicliques used to cover E^{free} do not cover any edges of E^{imp}. Then, the edges of E^{imp} should have been covered with at most $k - k_f = 2\ell + 2$ bicliques. Then, Lemma 3.44 implies that ψ is satisfiable.

Since $k = k_f + 2\ell + 2 = (4\log_2 n + 2\lceil \log_2 m \rceil + 6) + 2\log_2 n + 2 = \mathcal{O}(\log n)$, the theorem follows. □

3.7 Approximation Algorithms for Biclique Cover and Partition

In this section, we prove Theorems 3.14 and 3.15 by giving polynomial time approximation algorithms for BicCover, BicPart and EdgeCliqPart. First we give an algorithm that works for both BicCover and BicPart. The main constituent of this algorithm is the twin reduction described in Section 3.2.2.

Algorithm: Let G be the input graph. First we apply Reduction rule 3.17 (twin reduction) exhaustively on G. Let G' be the resulting graph. Assign each edge of G' arbitrarily to one of the end points. Let E_v be the set of edges assigned to vertex v. Note that if E_v is not empty, then it forms a star, which is a biclique. The set of all such star bicliques forms a biclique partition of G'. Let this biclique partition be $\mathcal{B}'$. Now, construct a biclique partition $\mathcal{B}$ of G with the same size as that of $\mathcal{B}'$ as given in Lemma 3.18. Note that $\mathcal{B}$ is also a biclique cover of G as any cover is also a partition.

Lemma 3.46. *The above algorithm correctly finds a biclique partition $\mathcal{B}$ of G (which is also a biclique cover of G) such that $|\mathcal{B}| \leq \frac{n}{\log_3 n} \cdot \mathsf{bc}(G) \leq \frac{n}{\log_3 n} \cdot \mathsf{bp}(G)$, where $n = |V(G)|$. In other words, the algorithm is a polynomial time approximation algorithm for* Bic Cover *and* BicPart*, giving an approximation ratio of $\frac{n}{\log_3 n}$.*

Proof. First let us prove the correctness, i.e, $\mathcal{B}$ is indeed a biclique partition of G. For this, it is sufficient to prove that $\mathcal{B}'$ is indeed a biclique partition of G'. It is clear that for each vertex v, if E_v is non-empty then it forms a star which is a biclique; moreover, each edge of G' is present in exactly one of the stars.

Next we prove the size bound on $\mathcal{B}$. By Lemma 3.18, $|\mathcal{B}| = |\mathcal{B}'|$. The number of bicliques in $\mathcal{B}'$ is at most $|V(G)|$. We know that $|V(G')| \leq 3^{\mathsf{bc}(G)}$ from Lemma 3.19. Hence,

$$|\mathcal{B}| = |\mathcal{B}'| \leq \min\{n, 3^{\mathsf{bc}(G)}\} = \frac{\min\{n, 3^{\mathsf{bc}(G)}\}}{\mathsf{bc}(G)} \cdot \mathsf{bc}(G) \leq \frac{n}{\log_3 n} \cdot \mathsf{bc}(G).$$

□

The above algorithm can be easily modified for the special case of bipartite graphs to achieve an approximation ratio of $n/\log_2 n$, where $n = \min(|L(G)|, |R(G)|)$. The only modification required in the algorithm is that while assigning edges to vertices, assign each edge to the smaller side (i.e. to the endpoint from $L(G)$ or $R(G)$ whichever is smaller). After twin-reduction, we know that $|L(G)|, |R(G)| \leq 2^{\mathsf{bc}(G)}$ from Lemma 3.19. Hence, it follows that the number of bicliques in the cover/partition that we output is at most $\frac{n}{\log_2 n} \cdot \mathsf{bc}(G)$. This completes the proof of Theorem 3.14.

We remark that for biclique partition, the above approximation algorithm can be extended to the weighted problem to get the same approximation ratio. This is because the twin reduction also works for weighted biclique partition. But this is not true for weighted biclique cover, and hence the approximation algorithm does not extend to weighted biclique cover.

We now give an approximation algorithm for EdgeCliqPart, achieving an approximation ratio of $\min((\mathsf{ecp}(G))^2, |V(G)|^{3/2}, |E(G)|^{2/3})$, and thereby proving Theorem 3.15. The main constituent of our algorithm is the kernelization described in Section 3.2.3, developed by Mujuni and Rosamond [125].

Algorithm: Let G be the input graph. First we apply Reduction rule 3.23 (deleting the included neighborhoods of simplicial vertices) exhaustively on G. Let r be the number of times we applied the rule. Let G' be the resulting graph. Picking each edge of G' as a clique gives an ECP $\mathcal{C}'$ of G'. Find a clique partition $\mathcal{C}$ of G of size $|\mathcal{C}'| + r$ as given by Lemma 3.25. Output $\mathcal{C}$.

Lemma 3.47. *The above algorithm correctly finds an ECP $\mathcal{C}$ of G such that $\frac{|\mathcal{C}|}{\mathsf{ecp}(G)} \leq$ $\min((\mathsf{ecp}(G))^2, |V(G)|^{4/3}, |E(G)|^{2/3})$*

Proof. Let $\alpha = \frac{|\mathcal{C}|}{\mathsf{ecp}(G)}$. It is clear that $\mathcal{C}'$ is an ECP of G'. Then by Lemma 3.25, we have that $\mathcal{C}$ is an ECP of G with size $|\mathcal{C}'| + r = |E(G')| + r$. We know $|E(G')| \leq (\mathsf{ecp}(G'))^3$ by Lemma 3.26. Also, we have $\mathsf{ecp}(G') = \mathsf{ecp}(G) - r$ by Lemma 3.25. Thus we have $|\mathcal{C}| \leq (\mathsf{ecp}(G) - r)^3 + r \leq (\mathsf{ecp}(G))^3$. Hence $\alpha \leq (\mathsf{ecp}(G))^2$. We also know $|\mathcal{C}| \leq |V(G)|^2$. Hence,

$$\begin{aligned} \alpha \leq & \min\left(\frac{|V(G)|^2}{\mathsf{ecp}(G)}, (\mathsf{ecp}(G))^2\right) \\ \implies \alpha \leq & |V(G)|^{4/3} \end{aligned}$$

Also, $|\mathcal{C}| \leq |E(G)|$. Hence,

$$\begin{aligned} \alpha \leq & \min\left(\frac{|E(G)|}{\mathsf{ecp}(G)}, (\mathsf{ecp}(G))^2\right) \\ \implies \alpha \leq & |E(G)|^{2/3} \end{aligned}$$

□

We remark that the above algorithm for EdgeCliqPart cannot be extended to the weighted case as the kernelization does not extend to the weighted case.

3.8 Open Problems

(1) Is there a polynomial (or even a sub-exponential) kernel for k-BipBicPart? Recall that the similar problem k-EdgeCliqPart has a k^2-kernel [125] (See section 3.2.3). This kernel is obtained by reducing simplicial vertices. A similar rule in the biclique case would be to reduce bisimplicial edges, but such a rule turns out to be not sound.

(2) We showed that one cannot get a $2^{2^{o(k)}}$-time algorithm for k-BicCover assuming ETH. But, can we get a constant (or even a log) factor approximation algorithm that runs in $\mathcal{O}^*(2^{poly(k)})$ time? The same question can be also asked for k-EdgeCliqCover.

(3) Is the $2^{\mathcal{O}(k^2)}$ dependence on k in the running time optimal for k-BipBicPart and k-BinRank($\mathbb{F}$). Can we improve it or otherwise can we show a matching lower bound? It is easy to see a lower bound of $2^{o(k)}$ assuming ETH, due to the reduction by Jiang and Ravikumar [87].

(4) Is it possible to extend the $\mathcal{O}^*(2^{k^2})$-algorithm for k-BipBicPart to k-BicPart?

(5) For the generalized ℓ_0-rank approximation problem, there is a PTAS known which runs in time $(\frac{1}{\varepsilon})^{2^{\mathcal{O}(k)}/\varepsilon^2}mn$ [67, 14]. The doubly exponential dependence on k cannot be improved in general as shown in [14]. But the instance used to prove this lower bound uses the arithmetic over boolean semi ring. In fact, they use our lower bound for biclique cover, i.e., Corollary 3.12 to prove this result. It is interesting whether one can get better dependence on k for arithmetic over a

field. In particular can we get a $2^{\text{poly}(k)}$ dependence? One way to do this will be to extend our ideas for the k-BinRank($\mathbb{F}$) algorithm to the approximate setting. That is, can we make use of linear dependence in some way to get better algorithm?

(6) We gave $\mathcal{O}(n/\log n)$-approximation algorithms for BicCover and BicPart that runs in polynomial time. Is it possible to shave of further log factors from the approximation factor? Note that for CliquePartition, where we want to partition the vertices into cliques, the best known polynomial time approximation algorithm has an approximation ratio of $\mathcal{O}(n(\log\log n)^2/\log^3 n)$ [81].

(7) It is not difficult to see that the 3^k-kernel for k-BicPart to k-BicWtdPart. This shows that k-BicWtdPart is in FPT. But the known kernels for k-BicCover (even the bipartite version), k-EdgeCliqCover and k-EdgeCliqPart does not seem to work with edge weights. At least, we could not find a way to make them work. To the best of our knowledge, these three edge-weighted versions are not even known to be in FPT. Neither are we aware of any W[1]-hardness for them. Hence, we ask the question, whether k-EdgeCliqWtdCover, k-EdgeCliqWtd Part, and k-BicWtdCover are in FPT?

CHAPTER 4

Hadwiger's Conjecture for Squares of 2-Trees

4.1 Introduction

The **Hadwiger number** of a graph G, denoted by $\boldsymbol{\eta(G)}$, is the largest integer t such that G contains a K_t-minor. The *four color theorem* is probably the most popular theorem in graph theory, and says that every planar graph can be 4-colored. In 1937, Wagner [159] proved that the four color theorem (which was only a conjecture then) is equivalent to the following statement: If a graph is K_5-minor free, then it is 4-colorable. In 1943, Hadwiger [80] proposed the following conjecture which is a far reaching generalization of the four color theorem.

Conjecture 4.1. *For any integer $t \geq 1$, every K_{t+1}-minor free graph is t-colorable; that is, $\eta(G) \geq \chi(G)$ for any graph G.*

Hadwiger's conjecture is well known to be a challenging problem. Bollobás, Catlin and Erdős [26] describe it as "one of the deepest unsolved problems in graph theory". Hadwiger himself [80] proved the conjecture for $t = 3$. (The conjecture is trivially true for $t = 1, 2$). In view of Wagner's result [159] mentioned above, Hadwiger's conjecture for $t = 4$ is equivalent to the four color theorem, the latter being proved by Appel and Haken [10, 11] in 1977. In 1993, Robertson, Seymour and Thomas [140] proved that Hadwiger's conjecture is true for $t = 5$. The conjecture remains unsolved for $t \geq 6$, though for $t = 6$ Kawarabayashi and Toft [92] proved that any graph that is K_7-minor free and $K_{4,4}$-minor free is 6-colorable.

Similar to other difficult conjectures in graph theory, attempting Hadwiger's conjecture for some natural graph classes may lead to new techniques and shed light on the general case. So far Hadwiger's conjecture has been proved for several classes of graphs, including line graphs [138], proper circular arc graphs [20], quasi-line graphs [49], 3-arc graphs [162], complements of Kneser graphs [163], and powers of cycles and their complements [106]. There is also an extensive body of work on the Hadwiger number; see, for example, [39] and [74].

Reed and Seymour [138] proved that Hadwiger's conjecture is true for line graphs. Recently, there have been multiple attempts to generalize this result to graph classes that properly contain all line graphs. This was typically achieved by identifying some features of line graphs and using them as defining properties of the super class. An important super class of line graphs introduced in [50], for which Hadwiger's conjecture has been proved [49], is the class of *quasi-line graphs*, which are graphs with the property that the neighborhood of every vertex can be partitioned into at most two cliques.

Our research started with an unsuccessful attempt to further generalize the above result by considering classes of graphs with the property that the neighborhood of every vertex can be partitioned into a small number of cliques. A natural choice for us was the class of square graphs of bounded degree graphs, where the *square* of a graph G, denoted by G^2, is the graph with the same vertex set as G such that two vertices are adjacent if and only if the distance between them in G is equal to 1 or 2. It is readily seen that in G^2, the neighborhood of each vertex v can be partitioned into at most $d_G(v)$ many cliques, where $d_G(v)$ is the degree of v in G. However, we soon realized that proving Hadwiger's conjecture for square graphs is as difficult as proving it for all graphs. This is true even for the squares of split graphs, where a graph is *split* if its vertex set can be partitioned into an independent set and a clique. This observation is our first result whose proof is straightforward and will be given in Section 4.3.

Theorem 4.2. *Hadwiger's conjecture is true for all graphs if and only if it is true for squares of split graphs.*

Since split graphs form a subclass of the class of chordal graphs, Theorem 4.2 implies:

Corollary 4.3. *Hadwiger's conjecture is true for all graphs if and only if it is true for squares of chordal graphs.*

Theorem 4.2 and Corollary 4.3 suggest that squares of chordal or split graphs may capture the complexity of Hadwiger's conjecture. These are curious results, though they may not make Hadwiger's conjecture easier to prove. Nevertheless, the availability of the property of being square of a split or chordal graph may turn out to be useful. Moreover, Theorem 4.2 motivates the study of Hadwiger's conjecture for squares of graphs. In particular, in light of Corollary 4.3, it would be interesting to study Hadwiger's conjecture for squares of some interesting subclasses of chordal graphs in the hope of getting new insights into the conjecture. As a step towards this, we prove that Hadwiger's conjecture is true for squares of a subclass of chordal graphs called 2-trees defined as follows.

Definition 4.4 (2-tree). *A* 2*-tree is a graph that can be constructed by beginning with the graph* K_2 *and applying the following operation a finite number of times: Pick an edge* $e = uv$ *in the current graph, introduce a new vertex* w*, and add edges* uw *and* vw *to the graph.*

The class 2-trees can be considered as the basic case of chordal graphs in the following sense. Chordal graphs are precisely the graphs that can be constructed by beginning with a clique and applying the following operation a finite number of times: Choose a clique in the current graph, introduce a new vertex, and make this new vertex adjacent to all vertices in the chosen clique. If we begin with a k-clique and choose a k-clique at each step, then we get the class of k*-trees.* The simplest case is when $k = 2$, i.e., the class of 2-trees.

We call a graph 2*-simplicial* if its vertices has an ordering such that the higher numbered neighbors of each vertex can be partitioned into at most 2 cliques. It can be easily verified that all quasi-line graphs are 2-simplicial graphs, but the converse is not true. Thus, in view of the above-mentioned result for quasi-line graphs [49], it would be interesting to study whether Hadwiger's conjecture is true for all 2-simplicial graphs. Considering the effort [49] required for quasi-line graphs, resolving Hadwiger's conjecture for 2-simplicial graphs is likely to be a difficult task. Moreover, the class of circular arc graphs is a proper subclass of 2-simplicial graphs [1] and as far as we know a lot of effort has already gone into proving Hadwiger's conjecture for circular arc graphs, without success. Therefore, before attempting the entire class of 2-simplicial graphs it seems rational to start with some different but interesting subclasses of 2-simplicial graphs. Viewing from the context of the squaring operation of graphs, we asked the following question: Is there a subclass of 2-simplicial graphs which can be expressed as the square of some natural class of graphs? If $u \in V(G)$ and $u_1, u_2, \ldots, u_t \in N_G(u) \cap H(u)$ (where $H(u)$ is the vertices of G that are higher numbered than u with respect to the 2-simplicial ordering), it is clear that in G^2, $\cup_i(H(u) \cap N_G[u_i])$ will be a subset of $N_{G^2}[u] \cap H(u)$. For each u_i, $N_G[u_i]$ will form a clique in G^2 but there is no reason why $N_{G^2}[u] \cap H(u)$ should be partitionable into at most two cliques, if $t \geq 3$. So, we are tempted to consider only squares of 2-degenerate graphs, since for 2-degenerate graphs, $t = |H(u) \cap N_G(u)| \leq 2$. Unfortunately, even squares of all 2-degenerate graphs are not 2-simplicial. If we carefully analyze the situation, we can see that if the two vertices in $H(u) \cap N_G(u)$ are adjacent to each other, the square of such a 2-degenerate graph will be a 2-simplicial graph. This subclass of 2-degenerate graphs is exactly the class of 2-trees. Note that though any 2-tree is a 2-degenerate graph, the converse is not always true. The square of any 2-tree is a 2-simplicial graph (but not necessarily a quasi-line graph), but the square of a 2-degenerate graph may not be a 2-simplicial graph. Thus 2-trees are a special class of 2-simplicial graphs that is not contained in the class of quasi-line graphs. We find squares of 2-trees to be one of the well-structured non-trivial cases to consider.

Our main result is the proof of Hadwiger's conjecture for 2-trees. We also prove an additional structural property about the branch sets (for the definition of a branch set of a minor, see Section 4.2) of the clique minor exhibiting the proof.

Theorem 4.5. *Hadwiger's conjecture is true for squares of* 2*-trees. Moreover, for any* 2*-tree T, T^2 has a clique minor of order $\chi(T^2)$ for which each branch set induces a path.*

Our result in fact holds for a superclass of 2-trees called **generalized 2-trees**. A graph is called a generalized 2-tree if it can be obtained by allowing one to join a new vertex to a clique of order 1 or 2 instead of exactly 2 in the above-mentioned construction of 2-trees. (This notion is different from the concept of a partial 2-tree which is defined as a subgraph of a 2-tree).

Corollary 4.6. *Hadwiger's conjecture is true for squares of generalized* 2*-trees. Moreover, for any generalized* 2*-tree G, G^2 has a clique minor of order $\chi(G^2)$ for which each branch set induces a path.*

in general, while proving hadwiger's conjecture for any class of graphs, it is also interesting to study the structure of the branch sets forming a clique minor of order no less than the chromatic number. Theorem 4.5 and Corollary 4.6 also provides this information for squares of 2-trees and generalized 2-trees respectively.

We remark that it is often challenging to establish Hadwiger's conjecture for squares of even very special classes of graphs. We elaborate this point for a few graph classes. Obviously, planar graphs form a super class of the class of 2-trees, but their squares

[1]Consider an ordering of the vertices of a circular arc graph such that a vertex u with a smaller arc always gets a smaller number.

seem to be much more difficult to handle than squares of 2-trees. In fact, the chromatic number of squares of planar graphs is a very well studied topic in the context of Wegner's conjecture [160]; we will say more about this in section 4.6. Another graph class related to 2-trees is the class of squares of 2-degenerate graphs. Recently, there was an attempt [16] to prove Hadwiger's conjecture for squares of a special class of 2-degenerate graphs, namely subdivision graphs. The *subdivision* of a graph G, denoted by $S(G)$, is obtained from G by replacing each edge by a path of length two. The square $S(G)^2$ of $S(G)$ is known as the total graph of G, and the chromatic number $\chi(S(G)^2)$ is simply the total chromatic number of G. Thus, unsurprisingly, Hadwiger's conjecture for squares of subdivision graphs is closely related to the long-standing total coloring conjecture, which can be stated as $\chi(S(G)^2) \leq \Delta(G) + 2$, where $\Delta(G)$ is the maximum degree of G. It was shown in [16] that Hadwiger's conjecture for squares of subdivisions is not difficult to prove if we assume that the total coloring conjecture is true. The best result to date for the total coloring conjecture, obtained by Reed and Molloy [124], asserts that $\chi(S(G)^2) \leq \Delta(G) + 10^{26}$. Using this result, it was proved in [16] that Hadwiger's conjecture is true for squares of subdivisions of highly edge-connected graphs. However, it seems non-trivial to prove Hadwiger's conjecture for squares of subdivisions of all graphs without getting tighter bounds for the total chromatic number.

In Section 4.2, we give some additional preliminary definitions and notations for the chapter. In Section 4.3, we prove Theorem 4.2. The proof of Theorem 4.5 is the main body of the chapter and will be given in Section 4.4. In Section 4.5 we prove Corollary 4.6 using Theorem 4.5. In Section 4.6, we make a few remarks and suggest some future directions to conclude the chapter.

4.2 Preliminaries

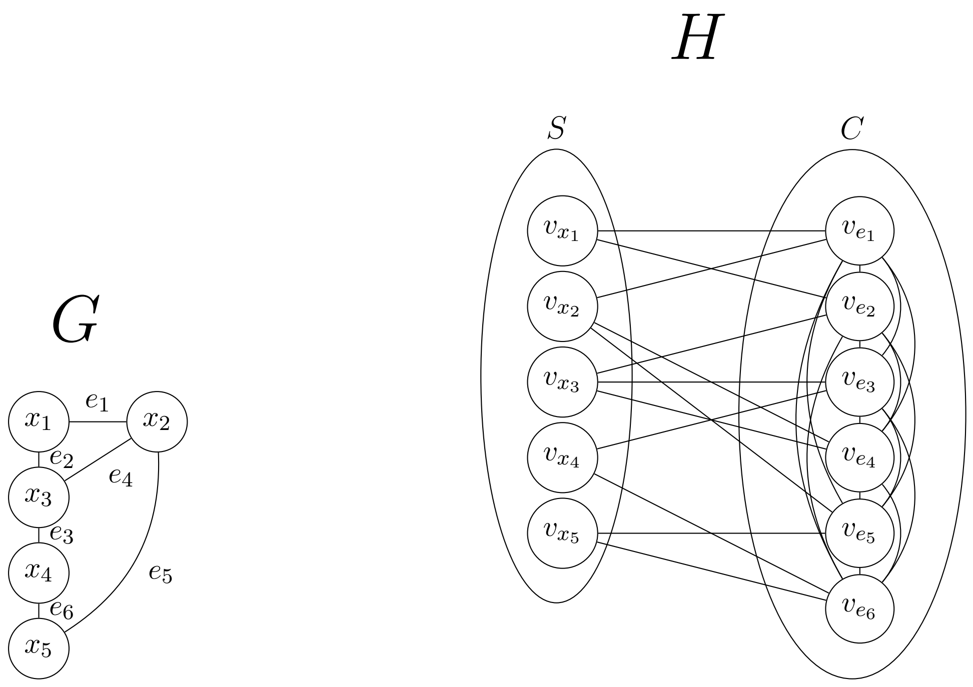
Recall that a graph H is called a **minor** of a graph G if a graph isomorphic to H can be obtained from a subgraph of G by contracting edges. An *H-minor* is a minor isomorphic to H, and a **clique minor** is a K_t-minor for some positive integer t, where K_t is the complete graph of order t. A graph is called *H-minor free* if it does not contain an H-minor. An H-minor of a graph G can be thought as a set of $t = |V(H)|$ vertex-disjoint subgraphs $G_1, \ldots, G_t$ of G such that each G_i is connected (possibly K_1) and the graph constructed in the following way is isomorphic to H: For each i, group all vertices of G_i to obtain a single vertex v_i, and add an edge between v_i and v_j if and only if there exists at least one edge of G between $V(G_i)$ and $V(G_j)$. The vertex set of each subgraph G_i is called a **branch set** of the minor H. This equivalent definition of a minor will be used throughout this chapter.

For a coloring μ of G and an $X \subseteq V(G)$, we define $\mu(X) := \{\mu(x) : x \in X\}$; and $|\mu|$ is defined as the number of colors used by μ.

4.3 Hadwiger's Conjecture and Squares of Split Graphs

In this section, we give the proof of Theorem 4.2. It suffices to prove that if Hadwiger's conjecture is true for squares of all split graphs then it is also true for all graphs.

So we assume that Hadwiger's conjecture is true for squares of split graphs. Let G be



(a) The graph G **(b)** The graph H

Figure 4.1: An example of the construction of the split graph H from a given graph G

an arbitrary graph with at least two vertices. Since, deleting isolated vertices does not affect the chromatic or Hadwiger number, without loss of generality we may assume that G has no isolated vertices. Construct a split graph H from G as follows (see Figure 4.1 for an example): For each vertex x of G, introduce a vertex v_x of H, and for each edge e of G, introduce a vertex v_e of H, with the understanding that all these vertices are pairwise distinct. Denote

$$S = \{v_x : x \in V(G)\}, \;\; C = \{v_e : e \in E(G)\}.$$

Construct H with vertex set $V(H) = S \cup C$ in such a way that no two vertices in S are adjacent, each pair of vertices in C are adjacent, and $v_x \in S$ is adjacent to $v_e \in C$ if and only if x and e are incident in G. Obviously, H is a split graph as its vertex set can be partitioned into the independent set S and the clique C.

Claim 1: The subgraph of H^2 induced by S is isomorphic to G.

In fact, for distinct $x, y \in V(G)$, v_x and v_y are adjacent in H^2 if and only if they have a common neighbor in H. Clearly, this common neighbor has to be from C, say v_e for some $e \in E(G)$, but this happens if and only if x and y are adjacent in G and $e = xy$. Therefore, v_x and v_y are adjacent in H^2 if and only if x and y are adjacent in G. This proves Claim 1.

Claim 2: In H^2 every vertex of S is adjacent to every vertex of C.

This follows from the fact that C is a clique of H and x is incident with at least one edge in G.

Claim 3: $\chi(H^2) = \chi(G) + |C|$.

In fact, by Claim 1 we may color the vertices of S with $\chi(G)$ colors by using an optimal coloring of G (that is, choose an optimal coloring ϕ of G and assign the color $\phi(x)$ to v_x for each $x \in V(G)$). We then color the vertices of C with $|C|$ other colors, one for each vertex of C. It is evident that this is a proper coloring of H^2 and hence $\chi(H^2) \leq \chi(G) + |C|$. On the other hand, since C is a clique, it requires $|C|$ distinct colors in any proper coloring of H^2. Also, by Claim 2 none of these $|C|$ colors can be assigned to any vertex of S in any proper coloring of H^2, and by Claim 1 the vertices of S need at least $\chi(G)$ colors in any proper coloring of H^2. Therefore, $\chi(H^2) \geq \chi(G) + |C|$ and Claim 3 is proved.

Claim 4: $\eta(H^2) = \eta(G) + |C|$.

To prove this claim, consider the branch sets of G that form a clique minor of G with order $\eta(G)$, and take the corresponding branch sets in the subgraph of H^2 induced by S. Take each vertex of C as a separate branch set. Clearly, these branch sets produce a clique minor of H^2 with order $\eta(G) + |C|$. Hence $\eta(H^2) \geq \eta(G) + |C|$.

To complete the proof of Claim 4, consider an arbitrary clique minor of H^2, say, with branch sets $B_1, B_2, \ldots, B_k$. Define $B'_i = B_i$ if $B_i \cap C = \emptyset$ (that is, $B_i \subseteq S$) and $B'_i = B_i \cap C$ if $B_i \cap C \neq \emptyset$. It can be verified that $B'_1, B'_2, \ldots, B'_k$ also produce a clique minor of H^2 with order k. Thus, if $k > \eta(G) + |C|$, then there are more than $\eta(G)$ branch sets among $B'_1, B'_2, \ldots, B'_k$ that are contained in S. In view of Claim 1, this means that G has a clique minor of order strictly bigger than $\eta(G)$, contradicting the definition of $\eta(G)$. Therefore, any clique minor of H^2 must have order at most $\eta(G) + |C|$ and the proof of Claim 4 is complete.

Since we assume that Hadwiger's conjecture is true for squares of split graphs, we have $\eta(H^2) \geq \chi(H^2)$. This together with Claims 3-4 implies $\eta(G) \geq \chi(G)$; that is, Hadwiger's conjecture is true for G. This completes the proof of Theorem 4.2.

4.4 Hadwiger's Conjecture and Squares of 2-Trees

In this section, we prove Theorem 4.5.

4.4.1 Prelude

Recall that a 2-tree is a graph that can be constructed by beginning with the graph K_2 and applying the following operation a finite number of times: Pick an edge $e = uv$ in the current graph, introduce a new vertex w, and add edges uw and vw to the graph. We call each application of this operation as a **step**. We say that e is **processed** in this step of the construction. We also say that w is a **vertex-child** of e; each of uw and vw is an **edge-child** of e; e is the **parent** of each of w, uw and vw; and uw and vw are **siblings** of each other. An edge e_2 is said to be an **edge-descendant** of an edge e_1, if either $e_2 = e_1$, or recursively, the parent of e_2 is an edge-descendant of e_1. A vertex v is said to be a **vertex-descendant** of an edge e if v is a vertex-child of an edge-descendant of e.

During the construction of a 2-tree, an edge e may be processed in more than one step. Suppose for edge e we make all the steps that process it consecutive by moving all the steps that process it to the position where it was processed first. It is not difficult to see that the resultant graph remains the same. So without loss of generality we may assume that for each edge e, all the steps in which e is processed occur consecutively.

We now define a **level** for each edge and each vertex of a 2-tree as follows. The level of the first edge, i.e., the only edge in the K_2 that we begin with, is defined to be 0. Also, the level of the end-vertices of this edge are also defined to be 0. Inductively, any vertex-child or edge-child of an edge with level k is said to have level $k+1$. Observe that two edges that are siblings of each other have the same level.

If there exists a pair of edges e, f with levels i, j respectively such that $i < j$ and the batch of consecutive steps where e is processed is immediately after the batch of consecutive steps where f is processed, then we can move the batch of steps where e is processed to the position immediately before the processing of f without changing the structure of the 2-tree. We repeat this procedure until no such pair of edges exists. So without loss of generality we may assume that the edges of level i are processed before edges of level j whenever $i < j$. We call this the **breadth-first processing**.

To prove Theorem 4.5, it is sufficient to prove that for any 2-tree T, the graph T^2 has a clique minor of order $\chi(T^2)$ such that each branch set of the clique minor induces a path. In the simplest case where $\chi(T^2) = 2$, T^2 contains a K_2-minor as there is at least one edge in T^2. Moreover, both branch sets of this K_2-minor are singletons (and hence induces path $\mathcal{P}_1$).

Hence it only remains to give the proof for a 2-tree T with $\chi(T^2) \geq 3$. Let T be a 2-tree with $\chi(T^2) \geq 3$. Denote by T_i, the 2-tree obtained after the i^{th} step in the construction of T as described above. Then there is a unique positive integer i^* such that $\chi(T^2) = \chi(T^2_{i^*})$ and $\chi(T^2_{i^*}) = \chi(T^2_{i^*-1}) + 1$. Define

$$G := T_{i^*}.$$

We will prove that $\eta(G^2) \geq \chi(G^2)$ and G^2 has a clique minor of order $\chi(G^2)$ for which each branch set induces a path. Once this is achieved, we then have $\eta(T^2) \geq \eta(G^2) \geq \chi(G^2) = \chi(T^2)$ and T^2 contains a clique minor of order $\chi(T^2)$ whose branch sets induce paths, as required to complete the proof of Theorem 4.5.

Denote by $\ell_{\texttt{max}}$ the maximum level of any edge of G. Then the maximum level of any vertex in G is also $\ell_{\texttt{max}}$. Observe that the level of the last edge processed is $\ell_{\texttt{max}} - 1$, and none of the edges with level $\ell_{\texttt{max}}$ has been processed at the completion of the i^*-th step, due to the breadth-first processing of edges. Obviously, $\ell_{\texttt{max}} \leq i^*$.

If $\ell_{\texttt{max}} \leq 1$, then G^2 is a complete graph and so $\chi(G^2) = \omega(G^2) = \eta(G^2)$. Moreover, G^2 contains a clique minor of order $\chi(G^2)$ for which each branch set induces the path $\mathcal{P}_1$. Hence the statement is true when $\ell_{\texttt{max}} = 0$ or 1. Hence, we assume $\ell_{\texttt{max}} \geq 2$ in the rest of the proof.

We will prove a series of lemmas that will be used in the proof of Theorem 4.5. See Figures 4.2 and 4.3 for the dependencies among these lemmas.

4.4.2 Pivot coloring, pivot vertex and its proximity

In this section, first we will fix a special coloring and vertex of the graph G called the pivot coloring and pivot vertex, denoted by μ and p respectively. Then, we will prove some lemmas showing the structure of graph G in the proximity of the vertex p in connection to the coloring μ. These structural insights will be used in the later sections

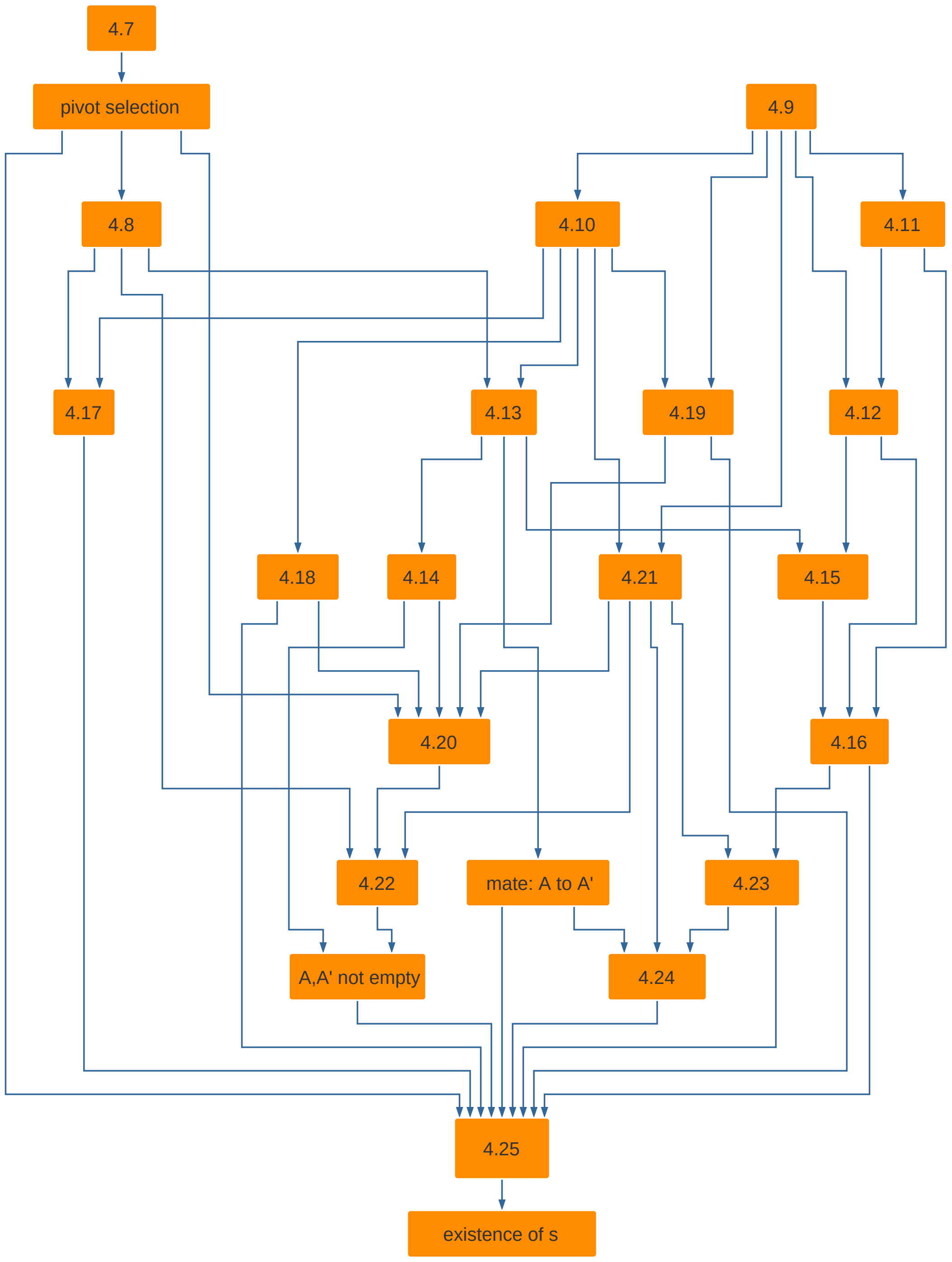


Figure 4.2: A flowchart of the proof of Theorem 4.5. This figure shows the part in Section 4.4.2 (proximity of pivot vertex). See Figure 4.3 for the remaining part. A node with label 4.n denotes Lemma/Corollary 4.n.

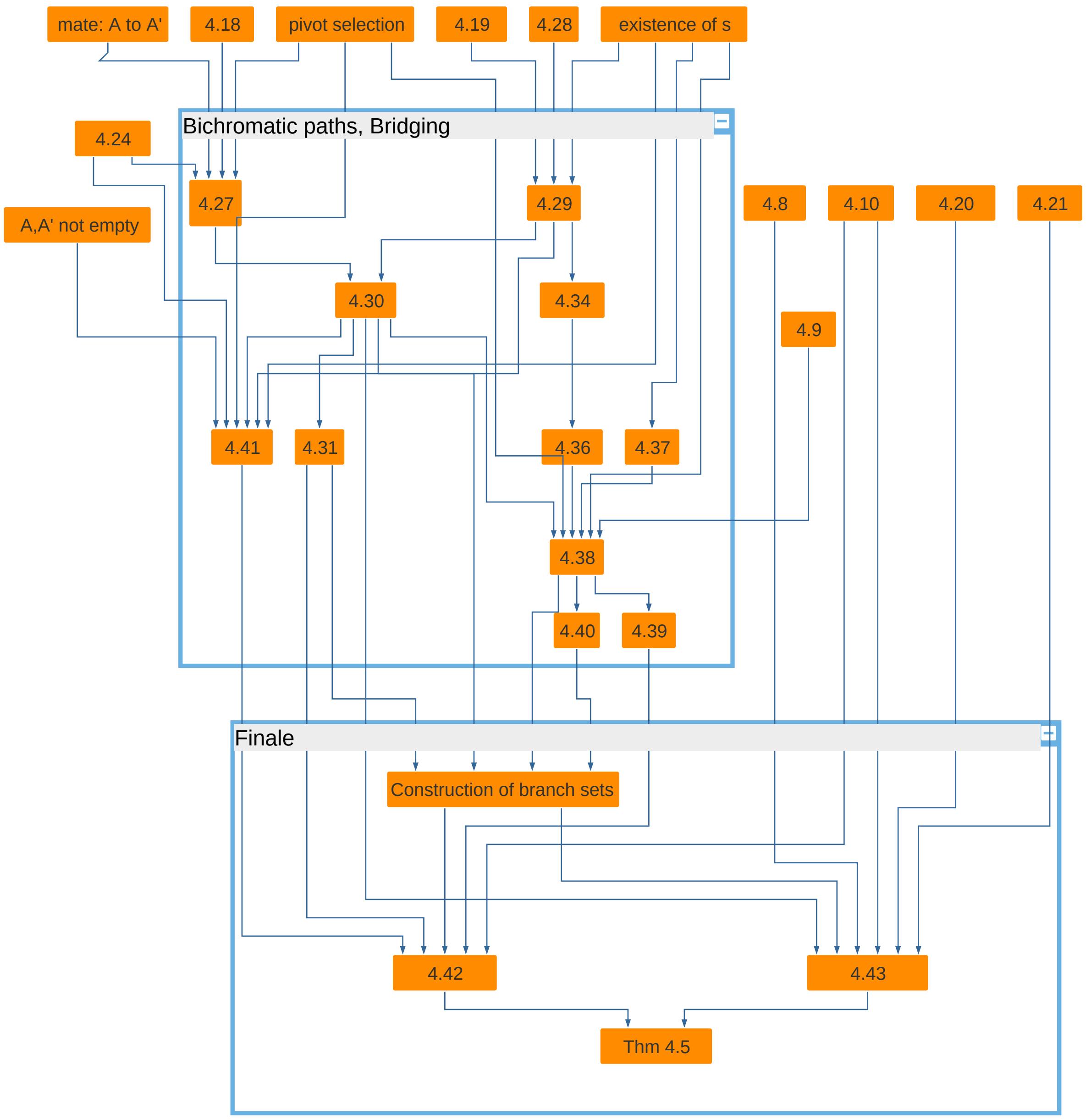


Figure 4.3: A flowchart of the proof of Theorem 4.5. This figure shows the part in Sections 4.4.3 (bichromatic paths), 4.4.4 (bridging), 4.4.5 (finale) and also the relationship with previous part (shown in Figure 4.2). A node with label 4.n denotes Lemma/Corollary 4.n

to construct the branch sets of the required clique minor. The relations between lemmas in this section is shown in Figure 4.2. We will define some vertex subnets around p in the process of proving the structural properties. Figure 4.4 shows an illustration of the different vertex sets that we will use. In Section 4.4.6, we give a list of the definitions of different sets and also some relations between them for easy reference.

First, we will fix the pivot coloring and pivot vertex. Towards this we prove the following lemma.

Lemma 4.7. *There exist an optimal coloring μ' of G^2 and a vertex p' of G at level $\ell_{\texttt{max}}$ such that p' is the only vertex with color $\mu'(p')$.*

Proof. Let p' be the vertex introduced in the step i^*. Then p' has level $\ell_{\texttt{max}}$. By the definition of $G = T_{i^*}$, there exists a proper coloring of $T^2_{i^*-1}$ using $\chi(G^2) - 1$ colors. Extend this coloring to G^2 by assigning a new color to p'. Let the resultant coloring be μ'. It is easy to see that μ' is an optimal coloring of G^2 under which p' has a unique color. □

Note that the pair (μ', p') in the proof above was just for exhibiting such a pair. There may be many candidates for (μ', p') satisfying the property in Lemma 4.7. We select one such pair as the pivot coloring and pivot vertex (μ, p) as follows.

Selection of pivot coloring and pivot vertex $(\boldsymbol{\mu}, \boldsymbol{p})$: Out of all the (coloring,vertex) pairs (μ', p') that satisfy the property in Lemma 4.7, we select a pair (μ, p) such that the minimum level among the vertices in $N(p)$ is as large as possible; we call μ the *pivot coloring* and p the *pivot vertex*.

From now on, when we say the color of a vertex, we mean the color under the coloring μ unless stated otherwise.

We define the following in the proximity of p (see Figure 4.4 for an illustration):

- $uw :=$ the parent of p;
- $t :=$ the vertex such that w is a child of edge ut. Note that the level of ut is $\ell_{\texttt{max}} - 2$, and uw and wt are siblings with level $\ell_{\texttt{max}} - 1$. The existence of t is ensured by the fact that $\ell_{\texttt{max}} \geq 2$;
- $B :=$ the set of vertex-children of wt;
- $C :=$ the set of vertex-children of uw. Note that $p \in C$.

It is clear from the definitions that the vertex sets B, C and $\{u, w, t\}$ are all distinct from each other.

Lemma 4.8. *All colors used by μ are present in $N^2[p]$. In other words, $|\mu(N^2[p])| = \chi(G^2)$.*

Proof. We know that p is the only vertex with color $\mu(p)$ under μ from the selection of μ and p. Hence, if there is a color c used by μ that is not present in $N^2[p]$, then we can recolor p with c to obtain a proper coloring of G^2 with $\chi(G^2) - 1$ colors, which is a contradiction. □

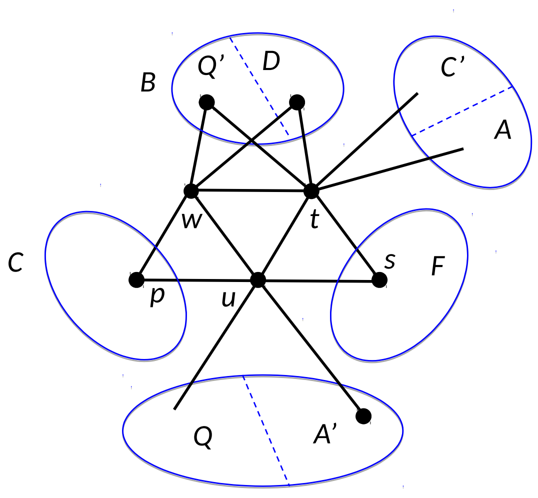


Figure 4.4: Vertex subsets of the 2-tree G around pivot vertex p. Note that there are possible edges that have not been shown in the figure. A summary of the definitions of the subsets and relations between them are given in Section 4.4.6 for easy reference.

Lemma 4.9. *For any vertex $b \in B$, its neighborhood $N(b) = \{w, t\}$ and for any $c \in C$, its neighborhood $N(c) = \{u, w\}$.*

Proof. Since both bw and bt have level $\ell_{\texttt{max}}$, they have not been processed at the time of completion of the i^*-th step. Hence the only neighbors of b are w and t. Similarly both cw and ct have level $\ell_{\texttt{max}}$ and hence have not been processed at the time of completion of the i^*-th step. Hence the only neighbors of c are w and u. □

Lemma 4.10. *(1) $N(w) = \{u, t\} \cup B \cup C$.*

(2) $N^2[p] = N[u] \cup B$.

Proof. (1) It is clear from the definitions of the vertices that all the vertices in $\{u, t\} \cup B \cup C$ are neighbors of w. It only remains to prove that there are no other neighbors for w. Suppose for the sake of contradiction that there was a neighbor v of w that was not in $\{u, t\} \cup B \cup C$. Since v is not an endpoint of the parent edge ut of w, the only way v can be a neighbor is if it is a vertex descendant of wu or wt. But since the only vertex descendants of wu and wt are the vertex sets C and B respectively, we have that $v \in B \cup C$, which is a contradiction.

(2) From Lemma 4.9 we have $N^2[p] = N[u] \cup N[w]$ and from statement (1) of the current lemma we have $N[w] \setminus N[u] = B$. Hence $N^2[p] = N[u] \cup B$.

□

We now define the following vertex sets (see Figure 4.4):

$$F := (N(u) \cap N(t)) \setminus \{w\}$$

$$C' := \{x \in N(t) : \mu(x) \in \mu(C)\}$$

$$A := N(t) \setminus (B \cup F \cup C' \cup \{u, w\}).$$

Note that $\mu(C') \subseteq \mu(C)$ by the definition of C. Also note that there may exist edges between F and $A \cup C'$. The following lemma shows that these sets are disjoint from each other and also from the sets B and C.

Lemma 4.11. *The sets $B, C, \{u, w, t\}, F, C'$ and A are all disjoint from each other.*

Proof. The fact that B, C and $\{u, w, t\}$ are disjoint from each other follows directly from their definitions. From the definition of F, it is clear that $F \cap \{u, w, t\} = \emptyset$. The vertices in C are not adjacent to t due to Lemma 4.9 and hence $C \cap (F \cup C' \cup A) = \emptyset$. The vertices in B are not adjacent to u due to Lemma 4.9 and hence $B \cap F = \emptyset$. From the definition of A, it follows that $A \cap (B \cup F \cup C' \cup \{u, w, t\}) = \emptyset$. From the definition of C', it is clear that $t \notin C'$.

It only remains to prove that $C' \cap (B \cup \{u, w\} \cup F) = \emptyset$. Suppose that $C' \cap (B \cup \{u, w\} \cup F) \neq \emptyset$ for the sake of contradiction. Let c' be a vertex in $C' \cap (B \cup \{u, w\} \cup F)$. Since $c' \in C'$, we have $\mu(c') \in \mu(C)$. Let c be the vertex in C such that $\mu(c) = \mu(c')$. Since $(B \cup \{u, w\} \cup F) \subseteq N^2(c)$, we have that $c' \in N^2(c)$. Thus c' and c are adjacent in G^2 and have the same color. Then μ is not a proper coloring of G^2, which is a contradiction. □

Lemma 4.12. *The sets A, C' and B are each disjoint from $N[u]$.*

Proof. $B \cap N[u] = \emptyset$ by Lemma 4.9. Suppose there is a vertex $v \in (A \cup C') \cap N[u]$ for the sake of contradiction. By the definition of A and C', we have $v \in N(t)$. If v is also in $N[u]$, then by definition of F, we have that $v \in F \cup \{u\}$. But by Lemma 4.11, the sets $A \cup C'$ and $F \cup \{u\}$ are disjoint, thus giving a contradiction. Hence $A \cap N[u] = \emptyset$ and $C' \cap N[u] = \emptyset$. □

Lemma 4.13. $\mu(A) \subseteq \mu(N(u) \setminus (C \cup F \cup \{w, t\}))$.

Proof. Let $a \in A$. Clearly, $\mu(a) \notin \mu(N^2(a))$. On the other hand, $\mu(a) \in \mu(N^2[p])$ by Lemma 4.8. So $\mu(a) \in \mu(N^2[p] \setminus N^2(a))$. Since

$$\begin{aligned} N^2[p] \setminus N^2(a) &= (N[u] \cup B) \setminus N^2(a) && \text{(using Lemma 4.10 (2))} \\ &\subseteq N(u) \setminus (F \cup \{w, t\}), \end{aligned}$$

it follows that $\mu(a) \in \mu(N(u) \setminus (F \cup \{w, t\}))$. Also, $\mu(a) \notin \mu(C)$ because, if $\mu(a) \in \mu(C)$ then $a \notin N(t)$ (otherwise a is in C' by the definition of C' and hence not in A by the definition of A, a contradiction) and hence $a \notin A$ by the definition of A, giving a contradiction. Therefore, $\mu(a) \in \mu(N(u) \setminus (C \cup F \cup \{w, t\}))$. □

By Lemma 4.13, for each color $c \in \mu(A)$, there is a c-colored vertex in $N(u) \setminus (C \cup F \cup \{w, t\})$. On the other hand, no two vertices in $N(u)$ can have the same color. So each color in $\mu(A)$ is used by exactly one vertex in $N(u)$. Let

$$A' := \{x \in N(u) : \mu(x) \in \mu(A)\}.$$

The following corollary follows directly from Lemma 4.13 and the definition of A'.

Corollary 4.14. $\mu(A') = \mu(A)$.

Lemma 4.15. *A' is disjoint from the sets $C, F, \{u, w, t\}, B, C'$ and A.*

Proof. From the definition of A', it is clear that $u \notin A'$. By Lemma 4.13 and the definition of A', we have that $A' \subseteq N(u) \setminus (C \cup F \cup \{u, w\})$. Thus A' is disjoint from C, F and $\{u, w\}$. By Lemma 4.12, we have that B, C' and A are disjoint from $N(u)$. Hence B, C' and A are disjoint from A'. □

Since no two vertices in A (A', respectively) are colored the same, the relation $\mu(a) = \mu(a')$ defines a bijection $a \mapsto a'$ from A to A', due to Corollary 4.14 and the definition of A'. We call a and a' the **mates** of each other and denote the relation by

$$a = \mathtt{mate}(a'), \;\; a' = \mathtt{mate}(a).$$

Note that $a \neq a'$ as A and A' are disjoint by Lemma 4.15.

Define

$$Q := N(u) \setminus (A' \cup C \cup F \cup \{w, t\}).$$

Then $\{A', Q\}$ is a partition of $N(u) \setminus (C \cup F \cup \{w, t\})$. Note that there may exist edges between F and $A' \cup Q$.

Define

$$Q' := \{x \in B : \mu(x) \in \mu(N(u))\}$$
$$D := B \setminus Q' = \{x \in B : \mu(x) \notin \mu(N(u))\}.$$

Lemma 4.16. *The sets $A', A, C, C', D, F, Q, Q', \{u, w, t\}$ are pairwise disjoint.*

Proof. The sets $A', A, B = Q' \sqcup D, C, C', F$ and $\{u, w, t\}$ are pairwise disjoint due to Lemmas 4.11 and 4.15. By definition, the set Q is disjoint from A', C, F and $\{u, w, t\}$. Since B, C' and A are disjoint from $N(u)$ by Lemma 4.12 and $Q \subseteq N(u)$ by definition of Q, we have that Q is disjoint from $B = Q' \sqcup D, C'$ and A. □

Lemma 4.17. *Suppose $D = \emptyset$. Then $\eta(G^2) \geq \chi(G^2)$. Moreover, $\chi(G^2) = \omega(G^2)$ and so G^2 contains a clique minor of order $\chi(G^2)$ for which each branch set is a singleton.*

Proof. We know $N^2[p] = N[u] \cup B$ by statement (2) of Lemma 4.10. Since $D = \emptyset$, we have $N^2[p] = N[u] \cup Q'$. Then by Lemma 4.8, all colors of μ are present in $N[u] \cup Q'$. But, $\mu(Q') \subseteq \mu(N[u])$ by the definition of Q'. Thus, all colors of μ are present in $N[u]$. Since $N[u]$ is a clique of G^2, it follows that $\chi(G^2) = |N[u]| \leq \omega(G^2)$. Therefore, $\chi(G^2) = \omega(G^2) \leq \eta(G^2)$. □

Due to Lemma 4.17, we know that the statement of the theorem holds in the case when $D = \emptyset$. Hence from now on we assume that $D \neq \emptyset$.

Lemma 4.18. *For any $d \in D$, no vertex in $N^2[p]$ other than d is colored $\mu(d)$.*

Proof. Suppose for the sake of contradiction that there is a vertex $v \in N^2[p] \setminus \{d\}$ such that $\mu(v) = \mu(d)$. Then for μ to be a proper coloring, $v \in N^2[p] \setminus N^2[d]$ We know that $N^2[p] = N[u] \cup B$ by Lemma 4.10 (2), and that $N^2[d] \supseteq B \cup (N[u] \setminus (Q \cup A'))$. Thus $N^2[p] \setminus N^2[d] \subseteq Q \cup A'$ and hence $v \in Q \cup A'$. Since $Q \subseteq N(u)$, we have $\mu(d) \notin \mu(Q)$ by the definition of D. Thus $v \notin Q$ and hence $v \in A'$. But $\mu(d) \notin \mu(A') = \mu(A)$ as $A \subseteq N^2[d]$. Hence $v \notin A'$, which is a contradiction. □

Lemma 4.19. *For any $b \in B$, $N^2[b] = A \cup B \cup F \cup C \cup C' \cup \{u, w, t\}$.*

Proof.

$$\begin{aligned} N^2(b) &= N[w] \cup N[t] && \text{(Using Lemma 4.9)} \\ &= (\{u,t\} \cup B \cup C) \cup (A \cup B \cup F \cup C' \cup \{u,w\}) && \text{(using Lemma 4.10 and} \\ & && \text{the definition of } A) \\ &= A \cup B \cup F \cup C \cup C' \cup \{u,w,t\}\,. \end{aligned}$$

□

Lemma 4.20. $\mu(Q) = \mu(Q')$.

Proof. We prove $\mu(Q') \subseteq \mu(Q)$ first. Suppose for the sake of contradiction that there exist a $q' \in Q'$ such that $\mu(q') \notin \mu(Q)$. By the definition of Q', $\mu(q') \in \mu(N(u))$. Thus $\mu(q') \in \mu(N(u)) \setminus \mu(Q) = \mu(C \cup \{w,t,s\} \cup A')$. But $\mu(q') \notin \mu(\{w,t,s\} \cup C)$ as $\{w,t,s\} \cup C \subseteq N^2(q')$. Thus $\mu(q') \in \mu(A')$. But since $\mu(A') = \mu(A)$ by Corollary 4.14, we get that $\mu(q') \in \mu(A) \subseteq \mu(N^2(q'))$, implying that μ is not a proper coloring of G^2, which is a contradiction.

Now we prove $\mu(Q) \subseteq \mu(Q')$. Suppose for the sake of contradiction that there exist a $q \in Q$ satisfying $\mu(q) \notin \mu(Q')$. Since $D \neq \emptyset$ (recall that we assumed so due to Lemma 4.17), we may take a vertex $d \in D$.

We claim that $\mu(q) \notin \mu(N^2(d))$. For the sake of contradiction, suppose otherwise, i.e., $\mu(q) \in \mu(N^2(d))$. Since μ is a proper coloring of G^2, we know that $\mu(q) \notin \mu(N^2(q))$. Thus $\mu(q) \in \mu\left(N^2(d) \setminus N^2(q)\right)$. We know from Lemma 4.19 that

$$N^2(d) \subseteq A \cup B \cup F \cup C \cup C' \cup \{u,w,t\}\,.$$

Also observe that

$$\begin{aligned} N^2[q] &\supseteq N[u] \\ &\supseteq \{u,w,t\} \cup F \cup C. \end{aligned}$$

Thus

$$\begin{aligned} N^2(d) \setminus N^2(q) &\subseteq A \cup B \cup C' \\ &= A \cup Q' \cup D \cup C'. \end{aligned}$$

Thus $\mu(q) \in \mu(A \cup Q' \cup D \cup C')$. But $\mu(q) \notin \mu(A) = \mu(A')$ as $A' \subseteq N^2(q)$; $\mu(q) \notin \mu(C') \subseteq \mu(C)$ as $C \subseteq N^2(q)$; $\mu(q) \notin \mu(Q')$ by our assumption; and $\mu(q) \notin \mu(D)$ by the definition of D. Hence we have a contradiction and hence $\mu(q) \notin N^2(d)$.

Since $\mu(q) \notin N^2(d)$, we can recolor d with $\mu(q)$ without violating the properness of the coloring in G^2. From Lemma 4.18, we know that d was the only vertex in $N^2[p]$ with color $\mu(d)$. But since we have recolored d with another color, we can now recolor p with $\mu(d)$ still maintaining a proper coloring of G^2. This new coloring does not use $\mu(p)$ as p was the only vertex having color $\mu(p)$ due to the property of the pivot vertex and now we have recolored p with another color. Thus the new coloring is proper and uses one less color than μ, implying that μ is not an optimal coloring of G^2, which is a contradiction. □

Lemma 4.21. $N^2[p] = B \cup Q \cup A' \cup F \cup \{u, w, t\} \cup C = N[w] \cup Q \cup A' \cup F$

Proof. We know $N^2[p] = N[w] \cup N[u]$ by Lemma 4.9. By Lemma 4.10, $N[w] = B \cup \{u, w, t\} \cup C$). Also, $N[u] = Q \cup A' \cup F \cup C \cup \{u, w, t\}$ from the definition of Q. Hence,

$$\begin{aligned} N^2[p] &= N[w] \cup N[u] \\ &= B \cup Q \cup A' \cup F \cup \{u, w, t\} \cup C \\ &= N[w] \cup Q \cup A' \cup F. \end{aligned}$$

□

Lemma 4.22. *Suppose $A = \emptyset$. Then $\eta(G^2) \geq \chi(G^2)$. Moreover, $\chi(G^2) = \omega(G^2)$ and hence G^2 contains a clique minor of order $\chi(G^2)$ for which each branch set is a singleton.*

Proof. From Lemma 4.21, we have that $N^2[p] = N[w] \cup Q \cup A' \cup F$. Since $A = \emptyset$, we have $A' = \emptyset$ by the definition of A. Hence $N^2[p] = N[w] \cup Q \cup F$. Since $\mu(Q) = \mu(Q')$ by Lemma 4.20 and $Q' \subseteq N[w]$, we have that $\mu(N^2[p]) = \mu(N[w] \cup F)$. By Lemma 4.8, we know $|\mu(N^2[p])| = \chi(G^2)$. On the other hand, $N[w] \cup F$ is a clique of G^2 and so $|\mu(N[w] \cup F)| \leq \omega(G^2)$. So $\chi(G^2) = |\mu(N^2[p])| = |\mu(N[w] \cup F)| \leq \omega(G^2)$, and therefore $\chi(G^2) = \omega(G^2) \leq \eta(G^2)$. □

Due to Lemma 4.22, we assume henceforth that $A \neq \emptyset$. This also implies that $A' \neq \emptyset$ using Corollary 4.14.

Lemma 4.23. $A \cap N^2[p] = \emptyset$.

Proof. We know $N^2[p] = B \cup Q \cup A' \cup F \cup \{u, w, t\} \cup C$ by Lemma 4.21 and that A is disjoint from $B \cup Q \cup A' \cup F \cup \{u, w, t\} \cup C$ due to Lemma 4.16. Hence, the lemma follows. □

Lemma 4.24. *For any $a' \in A'$, the vertex a' is the only vertex in $N^2[p]$ with color $\mu(a')$ under μ.*

Proof. Suppose for the sake of contradiction that there is a vertex $v \in N^2[p]$ such that $\mu(v) = \mu(a')$ but $v \neq a'$. Let $a = \mathtt{mate}(a')$.

$$\begin{aligned} N^2(a) \cup N^2[a'] &\supseteq B \cup Q \cup A' \cup F \cup \{u, w, t\} \cup C \\ &\supseteq N^2[p]. && \text{(by Lemma 4.21).} \end{aligned}$$

Hence $v \in N^2[\{a, a'\}]$. We already know $v \neq a'$. Also, $v \neq a$ as $A \cap N^2[p] = \emptyset$ by Lemma 4.23. Thus v is a vertex adjacent to either a or a' in G^2. But then μ is not a proper coloring as $\mu(v) = \mu(a') = \mu(a)$ (Recall that mates have the same color). Thus we have a contradiction. □

Lemma 4.25. *The following hold:*

(a) $\ell_{\mathtt{max}} \geq 3$;

(b) *the level of u is $\ell_{\mathtt{max}} - 2$.*

Proof. (a) Suppose $\ell_{\texttt{max}} \leq 2$ for the sake of contradiction. Recall that we have taken care of the case when $\ell_{\texttt{max}} \leq 1$ and hence have assumed $\ell_{\texttt{max}} \geq 2$. Thus $\ell_{\texttt{max}} = 2$. Then the level of ut is $\ell_{\texttt{max}} - 2 = 0$. This means ut is the unique edge with level 0. Moreover, $V(G) = N[\{u,t\}]$.

Take $a' \in A'$ and $d \in D$. (Recall that we have assumed $D, A, A' \neq \emptyset$ due to Lemmas 4.17 and 4.22). Let μ' be the coloring defined as follows: $\mu'(p) := \mu(a'), \mu'(a') = \mu(d)$, and for every $v \notin \{p, a'\}$, define $\mu'(v) := \mu(v)$. We know p is the only vertex in $N^2[p]$ with color $\mu(p)$ under μ by property of pivot vertex. Hence, the coloring μ' does not use color $\mu(p)$ by the construction of μ'. Since μ' does not use any additional color than μ, we have that μ' uses fewer colors than μ.
Case 1. μ' is a proper coloring:
Then μ is not an optimal coloring as it has more colors than μ'. Thus we have a contradiction.
Case 2. μ' is not a proper coloring:
Then there exist vertices v_1 and v_2 such that $\mu'(v_1) = \mu'(v_2)$ and $v_2 \in N^2(v_1)$. Since μ was a proper coloring and μ' differs from μ only in the colors of p and a', we can assume without loss of generality that $v_1 \in \{p, a'\}$.
Case 2.1 $v_1 = a'$:
Then $\mu'(v_2) = \mu'(v_1) = \mu'(a') = \mu(d)$ and $v_2 \in N^2(v_1) = N^2(a')$.
Case 2.1.1. $v_2 = d$:
Then $d \in N^2(a')$ and hence $a' \in N^2[d]$. By Lemma 4.19, we know $N^2[d] = A \cup B \cup F \cup C \cup C' \cup \{u, w, t\}$. Thus $a' \in A \cup B \cup F \cup C \cup C' \cup \{u, w, t\}$. But by Lemma 4.16, A' is disjoint from A, B, F, C, C' and $\{u, w, t\}$, and hence $a' \notin A \cup B \cup F \cup C \cup C' \cup \{u, w, t\}$, a contradiction.
Case 2.1.2. $v_2 \neq d$:
We have $v_2 \notin N[u] \subseteq N^2[p]$ because by Lemma 4.18, d is the only vertex in $N^2[p]$ with color $\mu(d)$. We also have $v_2 \notin N[t]$ for otherwise two distinct vertices (d and v_2) in $N[t]$ have the same color under μ, contradicting that μ is a proper coloring of G^2. Thus, $v_2 \notin N[u] \cup N[t] = V(G)$, a contradiction.
Case 2.2. $v_1 = p$:
Then $\mu'(v_2) = \mu'(v_1) = \mu'(p) = \mu(a')$ and $v_2 \in N^2(v_1) = N^2(p)$. By Lemma 4.24, a' is the only vertex in $N^2[p]$ with color $\mu(a')$. Hence $v_2 = a'$. Then $\mu'(v_2) = \mu'(a') = \mu(d)$. Since we already have $\mu'(v_2) = \mu(a')$, this should mean that $\mu(a') = \mu(d)$. Also, $\mu(\texttt{mate}(a)) = \mu(d)$ as mates have the same color in μ. But since $\texttt{mate}(a) \in N^2(d)$, μ is not a proper coloring. Thus we have a contradiction.

(b) Suppose for the sake of contradiction that level of u is not $\ell_{\texttt{max}} - 2$. Then, since the level of ut is $\ell_{\texttt{max}} - 2$, the level of t must be $\ell_{\texttt{max}} - 2$ and the level of u must be smaller than $\ell_{\texttt{max}} - 2$. Take any $d \in D$ (Recall that we have assumed $D \neq \emptyset$ due to Lemma 4.17). Denote by μ' the coloring obtained by starting from the coloring μ and exchanging the colors of d and p (while keeping the colors of all other vertices same). We will show that we would have selected (μ', d) as the pivot coloring and vertex instead of (μ, p) (see the selection of pivot coloring and pivot vertex in Section 4.4.2), thereby implying a contradiction. For this, first we prove that μ' is a proper coloring. By Lemma 4.18, we know that no vertex in $N^2[p]$ other than d is colored with $\mu(d)$ under μ. Also, by property of pivot vertex, we know that p is the only vertex with color $\mu(p)$ under μ.

Hence, by construction, μ' is a proper coloring of G^2. It is also an optimal coloring as μ' did not use any color that was not in μ. Observe that d is the only vertex with color $\mu'(d) = \mu(p)$ under the coloring μ'. The vertex with minimum level in $N(d)$ is t, whose level is $\ell_{\mathtt{max}} - 2$. The minimum level of a vertex in $N(p)$ is smaller than $\ell_{\mathtt{max}} - 2$ since the level of u is smaller than $\ell_{\mathtt{max}} - 2$. Then, we would have selected μ' and d as the pivot coloring and pivot vertex respectively instead of μ and p (see the selection of pivot coloring and pivot vertex in Section 4.4.2), which is a contradiction. □

Since $\ell_{\mathtt{max}} \geq 3$ by Lemma 4.25, there exist a a vertex $s \in F$ such that ut is a child of st. Note that the level of st is $\ell_{max} - 3$, and us is the sibling of ut and has level $\ell_{\mathtt{max}} - 2$.

4.4.3 Bichromatic paths

In this section, we introduce the concept of bichromatic paths. We will show the existence of some bichromatic paths in G^2. These bichromatic paths will be later crucial in the construction of branch sets. For proving the existence of bichromatic paths, we use a common technique in the area of graph coloring called Kempe chain (See [5] for example).

Definition 4.26. Given a proper coloring ϕ of G^2 and two distinct colors r and g, a path in G^2 is called a *(ϕ, r, g)-bichromatic path* if each of its vertices is colored r or g under the coloring ϕ.

Note that in a (μ, r, g)-bichromatic path, the vertices are colored alternatively with r and g as two vertices with same color cannot be adjacent.

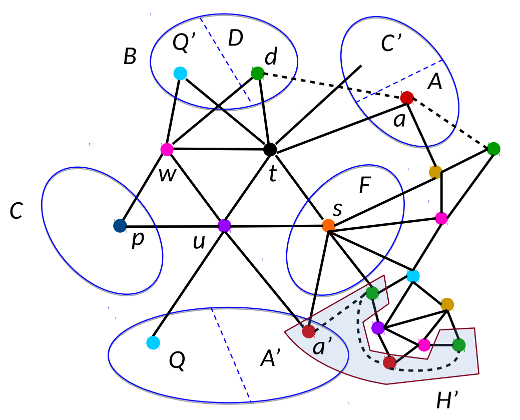
Lemma 4.27. *For any $a' \in A'$ and $d \in D$, there exists a $(\mu, \mu(a'), \mu(d))$-bichromatic path from a' to $\mathtt{mate}(a')$ in G^2.*

Proof. Let $a = \mathtt{mate}(a')$. Denote $r = \mu(a')$ $(= \mu(a))$ and $g = \mu(d)$. Then $r \neq g$ as $d \in N^2(a)$. Consider the subgraph H of G^2 induced by the set of vertices with colors r and g under μ. Let H' be the connected component of H containing a'. It suffices to show that a is contained in H'.

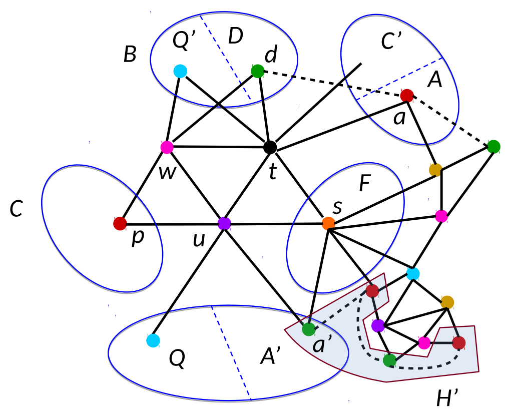
Suppose for the sake of contradiction that $a \notin V(H')$. Define

$$\mu'(v) = \begin{cases} \mu(v), & \text{if } v \in V(G) \setminus (V(H') \cup \{p\}) \\ r, & \text{if } v = p \\ r, & \text{if } v \in V(H') \text{ and } \mu(v) = g \\ g, & \text{if } v \in V(H') \text{ and } \mu(v) = r. \end{cases}$$

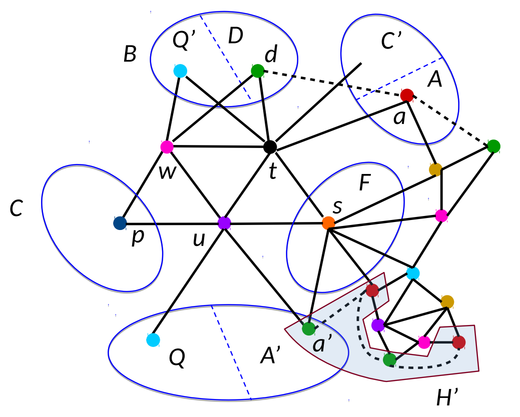
In particular, $\mu'(a') = g$. See Figure 4.5 for an illustration of the colorings.
Case 1. μ' is a proper coloring of G^2:
μ' does not use the color $\mu(p)$ as p was the only vertex that was colored with $\mu(p)$ under μ by property of pivot vertex. Hence μ' uses fewer colors than μ. This implies μ is not an optimal coloring, which is a contradiction.
Case 2. μ' is not a proper coloring of G^2:
Let μ'' be the coloring obtained by starting from the coloring μ and exchanging colors r and g within H' (see Figure 4.5(c)). It is easy to observe that μ'' is a proper coloring. Also, observe that μ' and μ'' differs only in the color of p. Thus, for μ' to be not a proper



(a) Coloring μ



(b) Coloring μ'



(c) Coloring μ''

Figure 4.5: An illustration of the colorings μ, μ' and μ'' used in the proof of Lemma 4.27. The dotted edges represent edges that are in G^2 but not in G. Note that we have not drawn all the edges of G and G^2.

coloring, there should be a vertex $v \in N^2(p)$ such that $\mu'(v) = \mu'(p) = r$.
Case 2.1. $v \in V(H')$:
In this case, we have $\mu(v) = g$. Then $v = d$ because d is the only vertex in $N^2[p]$ with color $\mu(d) = g$ under μ, using Lemma 4.18. But then a is in H' as a and d are adjacent in G^2 and $\mu(a) = r$. This is a contradiction to our assumption that $a \notin V(H')$.
Case 2.2. $v \notin V(H')$:
In this case, we have $\mu(v) = \mu'(v) = \mu'(p) = r$. By Lemma 4.24, we know that a' is the only vertex in $N^2[p]$ with color r. Hence $v = a' \in V(H')$, which is a contradiction. $\square$

Lemma 4.28. *For any edge $e = xy$ with level $\ell_{\mathtt{max}} - 2$ and any vertex-descendant z of e, we have $N^2(z) \subseteq N[\{x, y\}]$.*

Proof. Consider an arbitrary vertex v in $N^2(z)$. Since the level of e is $\ell_{\mathtt{max}} - 2$, there are only two possibilities for z. The first possibility is that z is a vertex-child of e. In this possibility, either v is a vertex-child of xz or yz, or $v \in \{x, y\}$, or $v \in N(x) \cup N(y)$; in each case we have $v \in N[\{x, y\}]$. The second possibility is that z is the vertex-child of an edge-child of e. Without loss of generality we may assume that z is the vertex-child of xq, where q is a vertex-child of e. Then either v is a vertex-child of yq or $v \in N[x]$; in each case we have $v \in N[\{x, y\}]$. $\square$

Lemma 4.29. *The following hold:*

(a) $N^2(A' \cup Q) \subseteq N[\{u, t, s\}]$;

(b) *if* $v \in N^2(A' \cup Q)$ *and* $\mu(v) \in \mu(B)$, *then* $v \in N(\{u, s\})$;

(c) *if* $v \in N^2(A' \cup Q)$ *and* $\mu(v) \in \mu(D)$, *then* $v \in N(s)$.

Proof. (a) Any vertex $x \in A' \cup Q$ is a vertex-descendant of ut or us. Since the levels of ut and us are both $\ell_{\mathtt{max}} - 2$, by Lemma 4.28, if x is a vertex-descendant of ut then $N^2(x) \subseteq N[\{u, t\}]$, and if x is a vertex-descendant of us then $N^2(x) \subseteq N[\{u, s\}]$. Therefore, $N^2(x) \subseteq N[\{u, t, s\}]$.

(b) Consider $v \in N^2(x)$ for some $x \in A' \cup Q$ such that $\mu(v) \in \mu(B)$. Since $v \in N[\{u, t, s\}]$ by (a), it suffices to prove $v \notin N[t]$. Suppose for the sake of contradiction that $v \in N[t]$.
Case 1. $v \notin B$:
Then for $\mu(v)$ to be in $\mu(B)$, there should be a $v' \in B$ such that $\mu(v') = \mu(v)$. But since both $v, v' \in N[t]$, this implies that μ is not a proper coloring, a contradiction.
Case 2. $v \in B$:
Then $N^2[v] = A \cup B \cup F \cup C \cup C' \cup \{u, w, t\}$ by Lemma 4.19. We know $A' \cup Q$ is disjoint from A, B, F, C, C', and $\{u, w, t\}$ by Lemma 4.16. Thus $x \in A' \cup Q$ is not in $N^2[v]$ and hence $v \notin N^2[x]$, a contradiction.

(c) By (b), every vertex $v \in N^2(A' \cup Q)$ with $\mu(v) \in \mu(D)$ must be in $N(\{u, s\})$ as $D \subseteq B$. If $v \in N(u)$, then $\mu(v) \in \mu(N(u))$ and so $\mu(v) \notin \mu(D)$ by the definition of D, a contradiction. Hence $v \notin N(u)$ and therefore $v \in N(s)$. $\square$

Define
$$D' := \{x \in N(s) : \mu(x) \in \mu(D)\}.$$

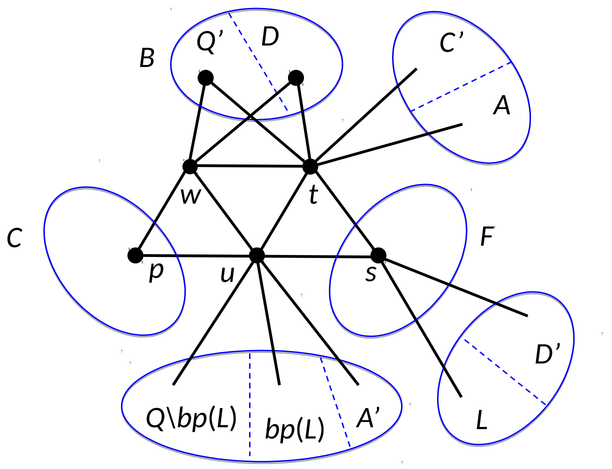


Figure 4.6: Vertex subsets used in the proof of Theorem 4.5. Note that there are possible edges that have not been shown in the figure. A summary of the definitions of the subsets and relations between them are given in Section 4.4.6 for easy reference.

Lemma 4.30. *The following hold:*

(a) $\mu(D') = \mu(D)$;

(b) *for each $a' \in A'$ and each $d' \in D'$, there exists a $(\mu, \mu(a'), \mu(d'))$-bichromatic path in G^2 from a' to $\mathtt{mate}(a')$ such that d' is adjacent to a' in this path.*

Proof. Observe that both statements of the lemma will follow if we can prove the following statement: For each $d \in D$, there exists $d' \in N(s)$ with $\mu(d') = \mu(d)$ such that for each $a' \in A'$, there exists a $(\mu, \mu(a'), \mu(d))$-bichromatic path from a' to $\mathtt{mate}(a')$ that contains the edge $a'd'$. Hence we will devote ourselves to proving this statement. Take any $d \in D$. For each $a' \in A'$, let $P_{a'}$ denote the $(\mu, \mu(a'), \mu(d))$-bichromatic path from a' to $\mathtt{mate}(a')$ guaranteed by Lemma 4.27. Let $d_{a'}$ be the vertex adjacent to a' in $P_{a'}$. Note that $d_{a'} \neq \mathtt{mate}(a')$ as $\mu(\mathtt{mate}(a')) = \mu(a')$. Clearly, $\mu(d_{a'}) = \mu(d)$. By Lemma 4.29(c), the vertex $d_{a'}$ is in $N(s)$. Thus $d_{a'} \in D'$. For distinct $a', a'' \in A'$, we have $d_{a'} = d_{a''}$ because otherwise $d_{a'}$ and $d_{a''}$ are adjacent in G^2 (as both are in $N(s)$) and have the same color ($\mu(d)$), thereby implying that μ is not a proper color. Thus there exists a $d' \in D'$ such that $d' = d_{a'}$ for each $a' \in A'$. Hence, for each $a' \in A'$, there exists a $(\mu, \mu(a'), \mu(d))$-bichromatic path from a' to $\mathtt{mate}(a')$ containing the edge $a'd'$. Thus we have proved the required statement. □

Since no two vertices in D (D', respectively) are colored the same, by Lemma 4.30 we have $|D| = |D'|$ and every $d' \in D'$ corresponds to a unique $d \in D$ such that $\mu(d) = \mu(d')$, and vice versa. We call d and d' the mates of each other, written as $d = \mathtt{mate}(d')$ and $d' = \mathtt{mate}(d)$. Note that for $a' \in A'$ and $d' \in D'$, $\mathtt{mate}(a')$ is adjacent to $\mathtt{mate}(d')$ in G^2. Lemma 4.30 also implies the following corollary.

Corollary 4.31. *The following hold:*

(a) *each $a' \in A'$ is adjacent to each $d' \in D'$ in G^2;*

(b) *for any $a' \in A'$ and $d' \in D'$, there exists a $(\mu, \mu(a'), \mu(d'))$-bichromatic path from d' to* $\texttt{mate}(d')$ *in G^2.*

4.4.4 Bridging sets, bridging sequences, and re-coloring

Here, we define the concepts of bridging sets and bridging sequences (see Figure 4.7(a) for an illustration) and prove some lemmas using them. These lemmas will be crucial for the construction of branch sets in the Finale (Section 4.4.5).

Definition 4.32. An ordered set $\{x_1, x_2, \ldots, x_k\}$ of vertices of G^2 is called a *bridging set* if for each i, $1 \leq i \leq k$, $x_i \in N(s) \setminus D'$ and there exists a vertex $q_i \in Q$ such that $\mu(q_i) = \mu(x_i)$ and q_i is not adjacent in G^2 to at least one vertex in $D' \cup \{x_1, x_2, \ldots, x_{i-1}\}$. Denote $q_i = \texttt{bp}(x_i)$ and call it the *bridging partner* of x_i. We also fix one vertex in $D' \cup \{x_1, x_2, \ldots, x_{i-1}\}$ not adjacent to q_i in G^2, denote it by $\texttt{bn}(q_i)$, and call it the *bridging non-neighbor* of q_i. (If there is more than one candidate, we fix one of them arbitrarily as the bridging non-neighbor.)

In the definition above we have $\texttt{bp}(x_i) \neq x_i$ for each i, for otherwise $\texttt{bp}(x_i)$ would be adjacent in G^2 to all vertices in $N(s)$ and so there is no candidate for the bridging non-neighbor of $\texttt{bp}(x_i)$, contradicting the definition of a bridging set.

In the following we take L to be a fixed bridging set with maximum cardinality. Note that $\mu(L) \subseteq \mu(Q)$ by the definition of a bridging set.

Definition 4.33. Given $z \in D' \cup L$, the *bridging sequence* of z is defined as the sequence of distinct vertices $s_1, s_2, \ldots, s_j$ such that $s_1 = z$, $s_j \in D'$, and for $2 \leq i \leq j$, s_i is the bridging non-neighbor of the bridging partner of s_{i-1}.

By definition of bridging set L, it is evident that the bridging sequence of every $z \in D' \cup L$ exists. In particular, for $d \in D'$, the bridging sequence of d consists of only one vertex, namely d itself.

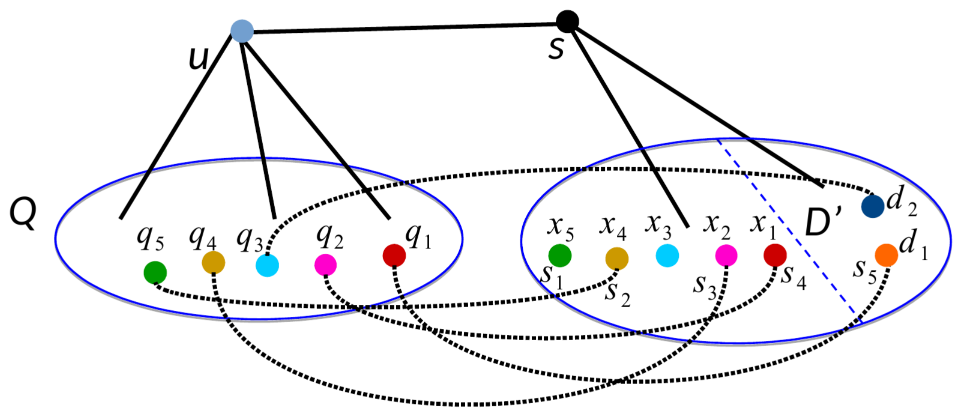
Lemma 4.34. *Let $x \in L$, $q = \texttt{bp}(x)$ and $y = \texttt{bn}(q)$. If there exists $v \in N^2(q)$ such that $\mu(v) = \mu(y)$, then $y \in L$ and $v = \texttt{bp}(y)$.*

Proof. We know that $q \in Q$ and $y \in D' \cup L$ from the definitions of bridging partner and bridging non-neighbor. Since $\mu(L) \subseteq \mu(Q)$ by the definition of a bridging set, we have $\mu(v) = \mu(y) \in \mu(D' \cup L) \subseteq \mu(B)$. Hence, by Lemma 4.29(b), v must be in $N(\{s, u\})$. If $v \in N(s)$, then $v = y$, but this cannot happen as $y = \texttt{bn}(q) \notin N^2(q)$. Hence $v \in N(u)$. This implies $\mu(v) \notin \mu(D) = \mu(D')$, from the definitions of D and D'. Therefore, $\mu(y) = \mu(v) \notin \mu(D')$. Since a bridging non-neighbor has to be from $D' \cup L$, we have that $y \in L$. Thus y has a bridging partner $\texttt{bp}(y)$ in Q with color $\mu(y)$. Since there can be only one vertex in $N(u)$ with color $\mu(y)$, we have that $v = \texttt{bp}(y)$. □

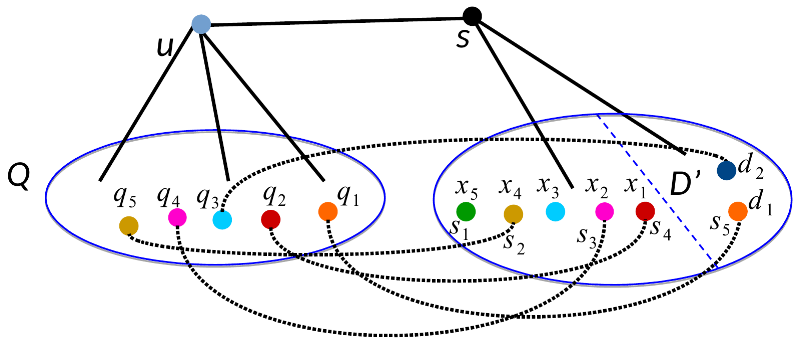
Definition 4.35. For any vertex $z \in D' \cup L$, we define it *bridging re-coloring* ψ_z as follows: Let $s_1, s_2, \ldots, s_j$ be the bridging sequence of z.

(a) $\psi_z(\texttt{bp}(s_i)) := \mu(s_{i+1})$ for $1 \leq i < j$ (Note that for $1 \leq i < j$, we have $s_i \in L$ and hence bridging partner is defined for s_i);

(b) $\psi_z(x) := \mu(x)$ for each $x \in V(G) \setminus \{\texttt{bp}(s_i) : 1 \leq i < j\}$.


u
s
Q
D'
q_5 q_4 q_3 q_2 q_1
x_5 x_4 x_3 x_2 x_1
s_1 s_2 s_3 s_4 s_5
d_2
d_1

(a) Bridging set $L = \{x_1, x_2, x_3, x_4, x_5\}$ and the bridging sequence s_1, s_2, s_3, s_4, s_5 of x_5. The dotted lines between two vertices represents that there is no edge between them in G^2. Here, $q_i = \mathtt{bp}(x_i)$ for all $i \in [5]$ and $\mathtt{bn}(q_1) = d_1, \mathtt{bn}(q_2) = x_1, \mathtt{bn}(q_3) = d_2, \mathtt{bn}(q_4) = x_2, \mathtt{bn}(q_5) = x_4$.


u
s
Q
D'
q_5 q_4 q_3 q_2 q_1
x_5 x_4 x_3 x_2 x_1
s_1 s_2 s_3 s_4 s_5
d_2
d_1

(b) The bridging recoloring ψ_{x_5}

Figure 4.7: Bridging set, bridging sequence, and bridging recoloring.

Observe that for $i_1 \neq i_2$ we have $\mu(s_{i_1}) \neq \mu(s_{i_2})$ as both $s_{i_1}, s_{i_2} \in N(s)$. So each color is used at most once for recoloring in (b) above.

Lemma 4.36. *For any $z \in D' \cup L$, ψ_z is an optimal coloring of G^2.*

Proof. Since ψ_z only uses colors of μ, it suffices to prove that it is a proper coloring of G^2. Let $s_1, s_2, \ldots, s_j$ be the bridging sequence of z. Suppose to the contrary that ψ_z is not a proper coloring of G^2. Then by the definition of ψ_z there exists $1 \leq i \leq j-1$ such that $\psi_z(\mathtt{bp}(s_i)) \in \psi_z(N^2(\mathtt{bp}(s_i)))$. Denote $x = \mathtt{bp}(s_i)$. Then there exists $v \in N^2(x)$ such that $\psi_z(v) = \psi_z(x) = \mu(s_{i+1})$. Observe x is the only vertex that has color $\mu(s_{i+1})$ under ψ_z but a different color under μ. Hence, we have $\mu(v) = \psi_z(v) = \mu(s_{i+1})$. Since $s_{i+1} = \mathtt{bn}(x)$, by Lemma 4.34 we have $s_{i+1} \in L$ and $v = \mathtt{bp}(s_{i+1})$. Since $s_{i+1} \in L$, we have that $i+1 \neq j$. Then, $\psi_z(v) = \psi_z(\mathtt{bp}(s_{i+1})) = \mu(s_{i+2}) \neq \mu(s_{i+1})$ by the definition of ψ_z, which is a contradiction. ☐

Lemma 4.37. *Let $a' \in A'$, $z \in L$, $r = \mu(a')$ and $g = \mu(z)$. Let $c \in \{r, g\}$. Then for any $x \in V(G)$, if $\psi_z(x) = c$ then $\mu(x) = c$.*

Proof. For the sake of contradiction, assume that there exist a vertex x such that $\psi_z(x) = c \in \{r, g\}$ but $\mu(x) \neq c$. Let $s_1, s_2, \ldots, s_j$ be the bridging sequence of z. From the construction of $\psi_z(x)$, it is clear that $x = \mathtt{bp}(s_i)$ and $\psi_z(x) = \mu(s_{i+1})$ for some $1 \leq i < j$. Then clearly $c = \psi_z(x) = \mu(s_{i+1})$.
Case 1. $c = r$:
Case 1.1. $i+1 < j$:
Then $s_{i+1} \in L$ and hence s_{i+1} has a bridging partner $\mathtt{bp}(s_{i+1})$. We know $\mathtt{bp}(s_{i+1}) \in Q$ and $\mu(\mathtt{bp}(s_{i+1})) = \mu(s_{i+1}) = r$ from the definition of bridging partner. But then a' and $\mathtt{bp}(s_{i+1})$ have the same color in μ, which is not possible as they are adjacent in G^2.
Case 1.2. $i+1 = j$:
Then $s_{i+1} \in D'$. Hence $\mu(s_{i+1}) \in \mu(D')$. Then $\mu(s_{i+1}) \in \mu(D)$ as $\mu(D') \subseteq \mu(D)$ by the definition of D'. Since every vertex in D is adjacent in G^2 to every vertex in A, we have that $\mu(s_{i+1}) \notin \mu(A)$. Then $\mu(s_{i+1}) \notin \mu(A')$ by the definition of A'. This is a contradiction as $\mu(s_{i+1}) = r = \mu(a') \in A'$.
Case 2. $c = g$:
But then s_{i+1} and s_1 has the same color in μ, which is not possible as they are adjacent in G^2 (because they have a common neighbor s in G). ☐

Lemma 4.38. *For any $a' \in A'$ and $z \in L$, there exists a $(\mu, \mu(a'), \mu(z))$-bichromatic path from a' to $\mathtt{mate}(a')$ in G^2 which contains the edge $a'z$.*

Proof. Let $r := \mu(a')$ and $g := \mu(z)$. In view of Lemma 4.37, it suffices to prove that there exists a (ψ_z, r, g)-bichromatic path from a' to a in G^2 that uses the edge $a'z$. Consider the subgraph H of G^2 induced by the set of vertices with colors r and g under ψ_z. Denote by H' the connected component of H containing a'. Let $a = \mathtt{mate}(a')$.
Case 1. $a \in V(H')$:
Since $a, a' \in V(H')$, there is a (ψ_z, r, g)-bichromatic path P from a' to a in G^2. It only remains to show that a' is adjacent to z in path P. Suppose for the sake of contradiction that the vertex adjacent to a' in P is not z but some other vertex v. Then $\psi_z(v) = g$, and by Lemma 4.37, $\mu(v) = g$. By Lemma 4.29(b), $v \in N(\{u, s\})$. Since $v \neq z$, we

have $v \notin N(s)$. Hence $v \in N(u)$, which implies $v = \texttt{bp}(z)$. Since $\psi_z(\texttt{bp}(z)) \neq g$ by the definition of ψ_z, it follows that $\psi_z(v) \neq g$, which is a contradiction.
Case 2. $a \notin V(H')$:
We will show that this is not possible by deriving a contradiction. Define a coloring ϕ of G^2 as follows:

$$\phi(v) = \begin{cases} \psi_z(v), & \text{if } v \in V(G) \setminus (V(H') \cup \{p\}) \\ r, & \text{if } v = p \\ r, & \text{if } v \in V(H') \text{ and } \psi_z(v) = g \\ g, & \text{if } v \in V(H') \text{ and } \psi_z(v) = r. \end{cases}$$

Case 2.1. ϕ is a proper coloring of G^2:
Recall that p is the only vertex in G with color $\mu(p)$ under μ by property of pivot vertex. By the definition of ψ_z, p remains to be the only vertex with color $\mu(p)$ under ψ_z. Hence ϕ uses one less color than ψ_z as it does not use the color $\psi_z(p) = \mu(p)$. This is a contradiction because, by Lemma 4.36, ψ_z is an optimal coloring of G^2.
Case 2.2. ϕ is not a proper coloring of G^2:
We know ψ_z is a proper coloring by Lemma 4.36. Observe that exchanging the two colors within $V(H')$ in the coloring ψ_z does not violate the properness of the coloring. Then for ϕ to be not a proper coloring, there exists a vertex $v \in N^2(p)$ such that $\phi(v) = r$.
Case 2.2.1. $v \in V(H')$:
Then $\psi_z(v) = g$. This implies $\mu(v) = g$ by Lemma 4.37.
Case 2.2.1.1. $v = \texttt{bp}(z)$:
Then $\psi_z(v) = \psi_z(\texttt{bp}(z)) = \mu(\texttt{bn}(\texttt{bp}(z))) \neq \mu(z) = g$, a contradiction.
Case 2.2.1.2. $v \neq \texttt{bp}(z)$:
Since $\mu(v) = g = \mu(\texttt{bp}(z))$, we have that $v \notin N^2[\texttt{bp}(z)] \supseteq N[u]$. Hence $v \in N^2[p] \setminus N[u]$. This implies $v \in B$ by Lemma 4.9(2).
Case 2.2.1.2.1. $v \in D$:
Then $\mu(v) = g \in \mu(D')$ due to Lemma 4.30(a). Let $d' \in D$ be such that $\mu(d') = g$. But then z and d' have the same color under μ and are adjacent in G^2 (as both are adjacent to s in G), thus implying μ is not a proper coloring, a contradiction.
Case 2.2.1.2.2. $v \in Q'$:
Then $a \in N^2(v)$. Hence, we have $a \in V(H')$ as a is colored with r and v is colored with g and both are adjacent in G^2. This is a contradiction to the fact that we are in Case 2.
Case 2.2.2. $v \notin V(H')$:
Then $\psi_z(v) = r$, and by Lemma 4.37, $\mu(v) = r$. The only vertex in $N^2[p]$ with color r under μ is a' by Lemma 4.24. Thus $v = a'$ and hence $\psi_z(v) = \psi_z(a') = g \neq r$, which is a contradiction. □

The following corollary is immediate from the above Lemma.

Corollary 4.39. *Each $a' \in A'$ is adjacent to each $z \in L$ in G^2.*

We now extend the definition of `mate` to the set L. Recall that $\mu(L) \subseteq \mu(Q)$ by the definition of a bridging set. This also means that $\mu(L) \subseteq \mu(Q')$ by Lemma 4.20. For each $q \in L$, define $\texttt{mate}(q)$ to be the vertex in Q' with the same color as q under the coloring μ. We now have the following corollary of Lemma 4.38.

Corollary 4.40. *For any $a' \in A'$ and $q \in L$, there is a $(\mu, \mu(a'), \mu(q))$-bichromatic path from q to $\texttt{mate}(q)$.*

Proof. Let $a := \texttt{mate}(a')$ and $q' := \texttt{mate}(q)$. From Lemma 4.38, we have a $(\mu, \mu(a'), \mu(q))$-bichromatic path $a'qv_1v_2 \dots v_ra$ in G^2. Since q' is adjacent to a in G^2 and $\mu(q') = \mu(q)$, we have that $qv_1v_2 \dots v_raq'$ is a $(\mu, \mu(a'), \mu(q))$-bichromatic path from q to q'. $\square$

Define

$$\texttt{bp}(L) := \{\texttt{bp}(q) : q \in L\}.$$

Then $\texttt{bp}(L) \subseteq Q$, $\mu(\texttt{bp}(L)) = \mu(L)$, and $\mu(L \cup (Q \setminus \texttt{bp}(L))) = \mu(Q) = \mu(Q')$. See Figure 4.6 for an updated picture of vertex sets around p including L and $\texttt{bp}(L)$.

Lemma 4.41. *For any $q \in Q \setminus \texttt{bp}(L)$, $D' \cup L \subseteq N^2[q]$.*

Proof. Suppose for the sake of contradiction that there exist a $q \in Q \setminus \texttt{bp}(L)$ and $z \in (D' \cup L)$ such that $z \notin N^2[q]$.
Case 1. $\mu(q) \in \mu(N(s))$:
Let $x \in N(s)$ be such that $\mu(q) = \mu(x)$. Then $x \neq q$ for otherwise $z \in N^2[q]$. Also, $x \notin L$ for otherwise, $\texttt{bp}(x)$ and q are adjacent in G^2 but have the same color under μ. We know that $\mu(x) \notin \mu(D)$ from the definition of D. Also $\mu(D) = \mu(D')$ by Lemma 4.30. Thus $\mu(x) \notin \mu(D')$ and hence $x \notin D'$. Now observe that $L \cup \{x\}$ is a larger bridging set than L by taking $\texttt{bp}(x) = q$ and $\texttt{bn}(q) = z$. This is a contradiction since L is a bridging set with maximum cardinality.
Case 2. $\mu(q) \notin \mu(N(s))$:
Since we have assumed $A' \neq \emptyset$ due to Lemma 4.22, we can take a vertex $a' \in A'$. Define a coloring ϕ of G^2 as follows:

$$\phi(v) := \begin{cases} \psi_z(v), & \text{if } v \in V(G) \setminus \{q, a', p\} \\ \psi_z(z), & \text{if } v = q \\ \psi_z(q), & \text{if } v = a' \\ \psi_z(a'), & \text{if } v = p \end{cases}$$

Let $c_1 := \phi(q) = \psi_z(z)$, $c_2 := \phi(a') = \psi_z(q)$ and $c_3 := \phi(p) = \psi_z(a')$. From the construction of ψ_z, we have that $c_1 = \psi_z(z) = \mu(z)$, $c_2 = \psi_z(q) = \mu(q)$ and $c_3 = \psi_z(a') = \mu(a')$. We know $\mu(q) \neq \mu(a')$ as q and a' are adjacent in G^2. Also, $\mu(q) \neq \mu(z) \in \mu(N(s))$ as we are in Case 2. We know $\mu(z) \in \mu(D' \cup L) \subseteq \mu(D \cup Q)$ (the latter part follows using Lemma 4.30 and the definition of L). Then $\mu(a') \neq \mu(z)$ as a' is adjacent in G^2 to every vertex in $D \cup Q$. Thus c_1, c_2, c_3 are all distinct from each other.

Recall that p is the only vertex in G with color $\mu(p)$ under μ by property of pivot vertex. By the definition of ψ_z, the vertex p remains to be the only vertex with color $\mu(p)$ under ψ_z. Hence ϕ uses one less color than ψ_z as it does not use the color $\psi_z(p) = \mu(p)$. Since by Lemma 4.36, ψ_z is an optimal coloring of G^2, ϕ cannot be a proper coloring of G^2. Hence one of the following three cases must happen.

Case 2.1: There exists $v \in N^2(q)$ such that $\phi(v) = \phi(q) = c_1$.

By the construction of ϕ, the vertex q is the only vertex with color $c_1 = \psi_z(z)$ under ϕ that has a different color under ψ_z. Therefore, since $v \neq q$, we have $\psi_z(v) = \phi(v) = c_1 = \mu(z)$. Since $\mu(z)$ is not a color that was recolored to some vertex during the construction of ψ_z, we have $\mu(v) = \mu(z)$. Then by Lemma 4.29(b), $v \in N(\{u, s\})$. If $v \in N(s)$, then $v = z$, which is a contradiction as $z \notin N^2[q]$. Thus, $v \in N(u)$, which implies $v = \texttt{bp}(z)$ as $\texttt{bp}(z)$ is the only vertex in $N(u)$ with color $\mu(z)$ under μ. However, then $\phi(v) = \phi(\texttt{bp}(z)) = \psi_z(\texttt{bp}(z)) = \mu(\texttt{bn}(\texttt{bp}(z))) \neq \mu(z) = c_1$, which is a contradiction.

Case 2.2: There exists $v \in N^2(a')$ such that $\phi(v) = \phi(a') = c_2$.

By the construction of ϕ, the vertex a' is the only vertex with color $c_2 = \psi_z(q)$ under ϕ that has a different color under ψ_z. Since $v \neq a'$, $\psi_z(v) = \phi(v) = c_2 = \mu(q)$. Since $\mu(q)$ is not a color that was recolored to some vertex during the construction of ψ_z, we have $\mu(v) = \psi_z(v) = \mu(q)$. By Lemma 4.29(b), $v \in N(\{u, s\})$. Since we are in Case 2, we have $\mu(q) \notin \mu(N(s))$ and hence $v \notin N(s)$. Thus $v \in N(u)$ which implies $v = q$. Then by the construction of ϕ, we have $\phi(v) = \phi(q) = c_1 \neq c_2$, a contradiction.

Case 2.3: There exists $v \in N^2(p)$ such that $\phi(v) = \phi(p) = c_3$.

By construction of ϕ, the vertex p is the only vertex with color $c_3 = \psi_z(a')$ under ϕ that has a different color under ψ_z. Since $v \neq p$, $\psi_z(v) = \phi(v) = c_3 = \mu(a')$. Since $\mu(a')$ is not a color that was recolored to some vertex during the construction of ψ_z, we have $\mu(v) = \mu(a')$. By Lemma 4.24, the vertex a' is the only vertex in $N^2(p)$ with color $\mu(a')$ under μ. This implies that $v = a'$. However, then $\phi(v) = \phi(a') = c_2 \neq c_3$, which is a contradiction. ☐

4.4.5 Finale

Denote by $a'_1, a'_2, \ldots, a'_k$ the vertices in A' and by $z_1, z_2, \ldots, z_\ell$ the vertices in $D' \cup L$, where $k = |A'|$ and $\ell = |D' \cup L|$. We will define a set $\mathcal{B}$ of branch sets. The set $\mathcal{B}$ will consists of a set of *singleton branch sets* $\mathcal{B}_1$ and a set $\mathcal{B}_p$ of *path branch sets.* Each branch set in $\mathcal{B}_1$ will be a single vertex and each branch set in $\mathcal{B}_p$ will form a path in G^2.

Case A: $k \leq \ell$ (See Figure 4.8(a) for an illustration).
Take $\mathcal{B}_1 := \{\{v\} : v \in N[w] \cup F\}$.
By Lemmas 4.30 and 4.38, for each $1 \leq i \leq k$, there is a $(\mu, \mu(a'_i), \mu(z_i))$-bichromatic path P_i from a'_i to $\texttt{mate}(a'_i)$. Take $\mathcal{B}_p := \{V(P_i) : 1 \leq i \leq k\}$.

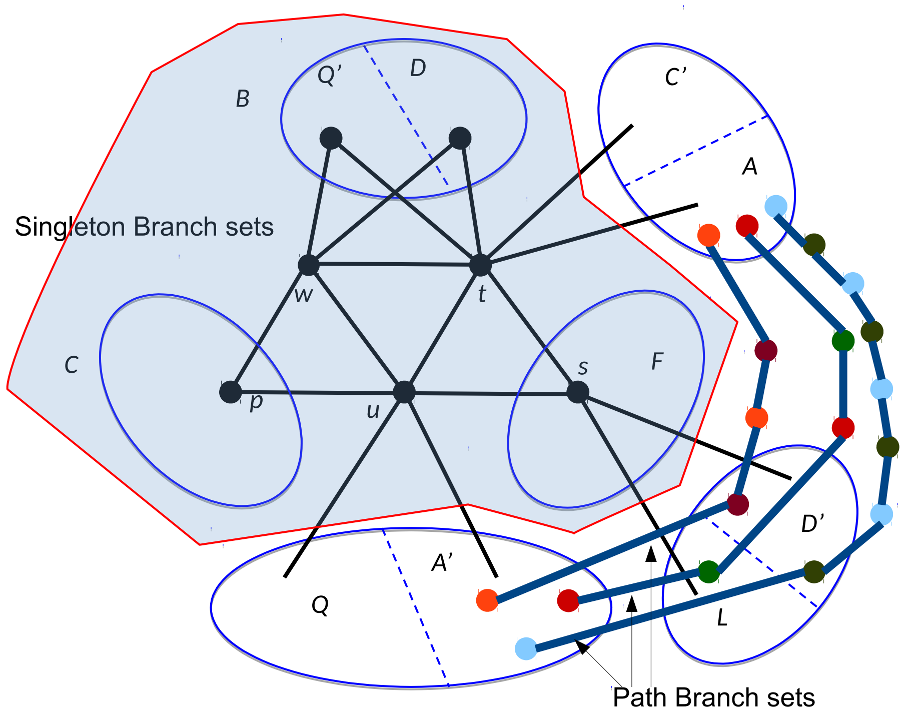
Case B: $\ell < k$ (See Figure 4.8(a) for an illustration).
Take $\mathcal{B}_1 := \{\{v\} : v \in N[u] \setminus \texttt{bp}(L)\}$.
By Corollaries 4.31(b) and 4.40, for each $1 \leq i \leq \ell$, there is a $(\mu, \mu(a'_i), \mu(z_i))$-bichromatic path P_i from z_i to $\texttt{mate}(z_i)$. Take $\mathcal{B}_p := \{V(P_i) : 1 \leq i \leq \ell\}$.

Note that the paths $P_1, P_2, \ldots$ are pairwise vertex-disjoint because the colors of the vertices in P_i and P_j are distinct for $i \neq j$. Therefore, the branch sets in $\mathcal{B}$ are pairwise disjoint.

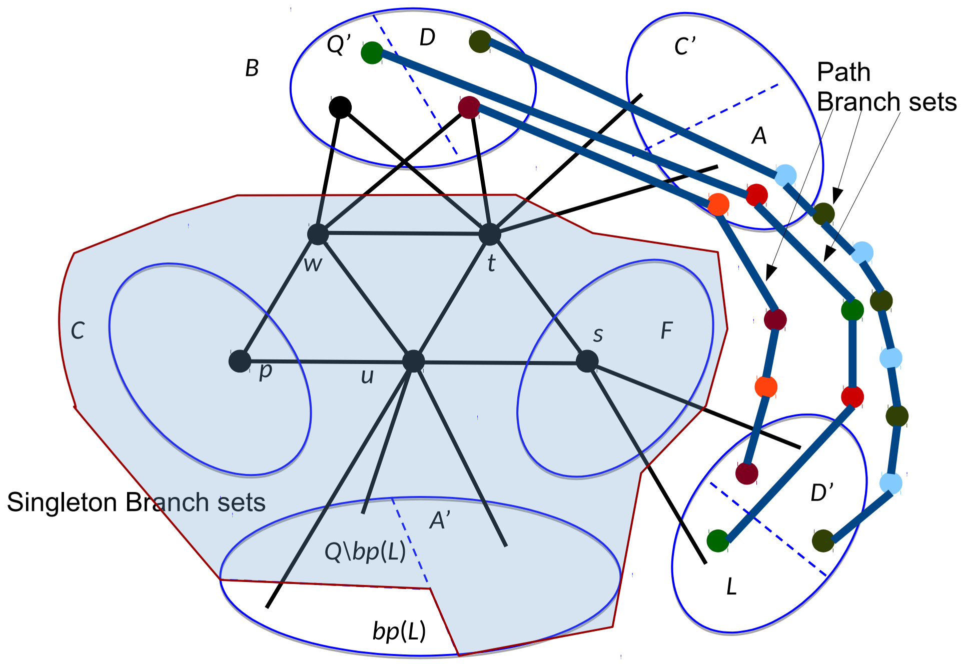
We can also assume that each P_i is a simple path because if it is not simple, we can find a subpath that is also the required bichromatic path. We can repeat this until we finally end up with a simple bichromatic path. Since we have that P_i is a simple path, we have that the graph induced by $V(P_i)$ in G^2 is a path.

Lemma 4.42. *Each pair of branch sets in $\mathcal{B}$ are joined by at least one edge in G^2.*

Proof. Consider Case A first. Recall that in Case A, the singleton branchsets $\mathcal{B}_1$ consists of vertices in $N[w] \cup F$. We know $N[w] = B \cup C \cup \{u, w, t\}$ from Lemma 4.10. From this, it can be easily observed that $N[w] \cup F$ is a clique of G^2. Hence the branch sets in $\mathcal{B}_1$ are pairwise adjacent. Also observe that for $1 \leq i \leq |A'|$, each vertex in $N[w] \cup F$ is adjacent to either a'_i or $\texttt{mate}(a'_i)$ in G^2. Hence each branch set in $\mathcal{B}_1$ is adjacent to each branch set in $\mathcal{B}_p$. For $1 \leq i, j \leq |A'|$ with $i \neq j$, we have $a'_j \in N^2[a'_i]$ and thus $V(P_i)$ and $V(P_j)$ are joined by at least one edge. Hence, the branch sets in $\mathcal{B}_p$ are pairwise adjacent. Thus, all the branch sets in $\mathcal{B}$ are pairwise adjacent.



(a) Case A of the Finale.



(b) Case B of the Finale.

Figure 4.8: Construction of Branch sets of the required clique minor

Now consider Case B. Recall that in Case B, the singleton branchsets $\mathcal{B}_1$ consists of vertices in $N[u] \setminus \texttt{bp}(L)$. Since $N[u] \setminus \texttt{bp}(L)$ forms a clique in G^2, the branch sets in $\mathcal{B}_1$ are pairwise adjacent. Now consider a z_i for some arbitrary $1 \leq i \leq |D' \cup L|$. We know $N[u] = Q \cup A' \cup C \cup F \cup \{u, w, t\}$ from the definition of Q. Since $\texttt{mate}(z_i) \in B$, it is adjacent in G^2 to all the vertices in $C \cup F \cup \{u, w, t\}$. By Corollaries 4.39 and 4.31(a), all vertices in A' are adjacent to z_i in G^2. By Lemma 4.41, all vertices in $Q \setminus \texttt{bp}(L)$ are adjacent to z_i in G^2. Thus, every vertex in $N[u] \setminus \texttt{bp}(L)$ is adjacent in G^2 to either z_i or $\texttt{mate}(z_i)$. Hence, each branch set in $\mathcal{B}_1$ is adjacent to each branch set in $\mathcal{B}_p$. Since $z_i \in N(s)$ for $1 \leq i \leq |D' \cup L|$, we have that z_i and z_j are adjacent in G^2 for all $1 \leq i \neq j \leq |D' \cup L|$. Hence, the branch sets in $\mathcal{B}_p$ are pairwise adjacent. Thus, all the branch sets in $\mathcal{B}$ are pairwise adjacent. $\square$

Lemma 4.43. $|\mathcal{B}| \geq \chi(G^2)$.

Proof. First let us consider Case A, i.e, when $|A'| \leq |D' \cup L|$. In this case,

$$\begin{aligned} |\mathcal{B}_1| &= |N[w] \cup F| \\ &= |B \cup C \cup \cup \{w, t, u\} \cup F| \qquad \text{(by Lemma 4.10)} \end{aligned}$$

Also, $|\mathcal{B}_p| = |A'|$ in this case. Hence,

$$\begin{aligned} |\mathcal{B}| &= |\mathcal{B}_1| + |\mathcal{B}_p| \\ &= |B \cup C \cup F \cup \{w, t, u\}\,| + |A'| \\ &\geq |B \cup C \cup F \cup \{w, t, u\} \cup A'| \end{aligned}$$

By Lemma 4.21, we know

$$B \cup C \cup F \cup \{w, t, u\} \cup A' = N^2[p] \setminus Q$$

Then,

$$\mu(B \cup C \cup F \cup \{w, t, u\} \cup A') = \mu(N^2[p] \setminus Q)$$

Since $\mu(Q) = \mu(Q') \subseteq \mu(B)$ by Lemma 4.20,

$$\begin{aligned} \mu(B \cup C \cup F \cup \{w, t, u\} \cup A') &= \mu\left(N^2\,[p]\right) \\ &= \chi(G^2) \qquad \text{(using Lemma 4.8).} \end{aligned}$$

Thus,

$$\begin{aligned} |\mathcal{B}| &= |B \cup C \cup F \cup \{w, t, u\} \cup A'| \\ &\geq |\mu\left(B \cup C \cup F \cup \{w, t, u\} \cup A'\right)| \\ &= \chi(G^2) \end{aligned}$$

Now consider Case B, i.e., when $|D' \cup L| < |A'|$. In this case,

$$\begin{aligned} |\mathcal{B}_1| &= |N[u] \setminus \texttt{bp}(L)| \\ &= |C \cup F \cup A' \cup (Q \setminus \texttt{bp}(L)) \cup \{w, t, u\}\,| \qquad \text{(by definition of } Q) \end{aligned}$$

Also, $|\mathcal{B}_p| = |D' \cup L|$ in this case. Hence,

$$\begin{aligned}
|\mathcal{B}| &= |\mathcal{B}_1| + |\mathcal{B}_p| \\
&= |C \cup F \cup A' \cup (Q \setminus \texttt{bp}(L)) \cup \{w,t,u\}\,| + |D' \cup L| \\
&\geq |C \cup F \cup A' \cup (Q \setminus \texttt{bp}(L)) \cup \{w,t,u\} \cup D' \cup L| \\
&\geq |\mu\,(C \cup F \cup A' \cup (Q \setminus \texttt{bp}(L)) \cup \{w,t,u\} \cup D' \cup L)\,|
\end{aligned}$$

We know $\mu(D') = \mu(D)$ by Lemma 4.30, $\mu(L) = \mu(\texttt{bp}(L))$ by the definition of bridging partner, and $\mu(Q') = \mu(Q)$ by Lemma 4.20. Therefore,

$$\begin{aligned}
|\mathcal{B}| &\geq |\mu\,(C \cup F \cup A' \cup Q' \cup \{w,t,u\} \cup D)\,| \\
&\geq |\mu(N^2[p])| && \text{(using Lemma 4.21)} \\
&= \chi(G^2) && \text{(using Lemma 4.8).}
\end{aligned}$$

□

Theorem 4.5 follows immediately from Lemmas 4.42 and 4.43.

4.4.6 A summary of different vertex sets used and relations between them

- $\mu, p :=$ the pivot coloring and the pivot vertex respectively. Out of all (coloring, vertex) pairs that satisfy Lemma 4.7, we select a pair (μ, p) such that the minimum level of the vertices in $N(p)$ is as large as possible;
- $uw :=$ the parent of p;
- $t :=$ the vertex such that w is a child of ut, so that the level of ut is $\ell_{\texttt{max}} - 2$, and uw and wt are siblings with level $\ell_{\texttt{max}} - 1$ (the existence of t is ensured by the fact that $\ell_{\texttt{max}} \geq 2$);
- $B :=$ the set of vertex-children of wt;
- $C :=$ the set of vertex-children of uw;
- $F := (N(u) \cap N(t)) \setminus \{w\}$;
- $C' := \{x \in N(t) : \mu(x) \in \mu(C)\}$;
- $\mu(C') \subseteq \mu(C)$;
- $A := N(t) \setminus (B \cup F \cup C' \cup \{u, w\})$;
- $A' := \{x \in N(u) : \mu(x) \in \mu(A)\}$;
- $\mu(A') = \mu(A)$;
- $Q := N(u) \setminus (A' \cup C \cup F \cup \{w, t\})$;
- $D := \{x \in B : \mu(x) \notin \mu(N(u))\}$;

- $Q' := B \setminus D$;
- $\mu(Q) = \mu(Q')$ unless $D = \emptyset$ (the case $D = \emptyset$ can be dealt with easily as shown in Lemma 4.17);
- $s :=$ the vertex in F such that ut is a child of st;
- $D' := \{x \in N(s) : \mu(x) \in \mu(D)\}$;
- $\mu(D') = \mu(D)$;
- $L :=$ A bridging set (see Definition 4.32) with maximum cardinality;
- $\mu(L) \subseteq \mu(Q)$.
- $\mu(\mathtt{bp}(L)) = \mu(L)$.

4.5 Hadwiger's Conjecture for Squares of Generalized 2-Trees

We now prove Corollary 4.6 using Theorem 4.5. We will use induction on the number of vertices. It can be easily verified that if G is a generalized 2-tree with small number of vertices, say at most 4, then G^2 has a clique minor of order $\chi(G^2)$ for which each branch set induces a path. Our induction assumption is that for some integer $n \geq 5$, for any generalized 2-tree H with at most $n-1$ vertices, H^2 has a clique minor of order $\chi(H^2)$ for which each branch set induces a path. Let G be a generalized 2-tree with n vertices. If G is a 2-tree, then by Theorem 4.5, the result in Corollary 4.6 is true for G^2. Assume that G is not a 2-tree. Then at some step in the construction of G, a newly added vertex v is made adjacent to a single vertex u in the existing graph. Then the vertices which are descendants of v or uv will never be adjacent to any vertices that were created before v except u. This means that u is a cut vertex of G. Thus G is the union of two edge-disjoint subgraphs G_1, G_2 with $V(G_1) \cap V(G_2) = \{u\}$. If $\chi(G^2) \leq N_G[u]$, then we have a clique in G^2 of size at least $\chi(G^2)$ as $N_G[u]$ is a clique in G^2. Then we have a clique minor with the branch sets as singletons and hence we are done. Hence, assume that $\chi(G^2) > N_G[u]$. Without loss of generality we may assume $\chi(G_1^2) \leq \chi(G_2^2)$. Observe that G^2 is the union of G_1^2, G_2^2, and the clique induced by $N_G[u]$ in G^2. Consider any optimal coloring c_1 of G_1^2 and c_2 of G_2^2. So, c_1 uses colors $1, 2, \ldots, \chi(G_1^2)$ and c_2 uses colors $1, 2, \ldots, \chi(G_2^2)$. Denote $N_i = N_{G_i}(u)$ for $i = 1, 2$. Then in c_i, the vertices in N_i need pairwise distinct colors. We can assume without loss of generality that both c_1 and c_2 uses color 1 for u (otherwise just swap the color used on u and 1 everywhere). Similarly, we can also assume without loss of generality that c_1 colors N_1 with colors $2, 3, \ldots, |N_1| + 1$ and and that c_2 colors N_2 with colors $\chi(G_2^2), \chi(G_2^2) - 1, \ldots, \chi(G_2^2) - |N_2| + 1$. Since $\chi(G^2) > N_G[u] = |N_1| + |N_2| + 1$ and $\chi(G_2^2) > \chi(G_1^2)$, we have that the colors $1, 2, 3, \ldots, |N_1| + 1, \chi(G_2^2), \chi(G_2^2) - 1, \ldots, \chi(G_2^2) - |N_2| + 1$ are all distinct with each other. Hence the coloring c where we take $c(v) = c_1(v)$ if $v \in V(G_1)$ and $c(v) = c_2(v)$ if $v \in V(G_2)$ is a proper coloring of G^2. Also, observe that c uses only $\chi(G_2^2)$ many colors. Since G_2 is a generalized 2-tree, by the induction

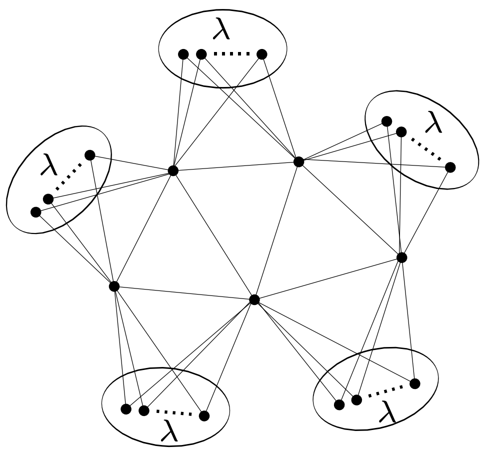


Figure 4.9: A 2-tree G with $\omega(G^2) = 2\lambda + 5$ and $\chi(G^2) = 3\lambda + 3$.

hypothesis, G_2^2 has a clique minor of size $\chi(G_2^2)$ for which each branch set induces a path. This clique minor then gives the required clique minor for G^2.

4.6 Concluding remarks and Open Problems

We have proved that for any 2-tree G, G^2 has a clique minor of order $\chi(G^2)$. Since large cliques played an important role in our proof of this result, it is natural to ask whether G^2 has a clique of order close to $\chi(G^2)$, say, $\omega(G^2) \geq c\chi(G^2)$ for a constant c close to 1 or even $\omega(G^2) = \chi(G^2)$. Since the class of 2-trees contains all maximal outerplanar graphs, this question seems to be relevant to Wegner's conjecture [160], which asserts that for any planar graph G with maximum degree Δ, $\chi(G^2)$ is bounded from above by 7 if $\Delta = 3$, by $\Delta + 5$ if $4 \leq \Delta \leq 7$, and by $(3\Delta/2) + 1$ if $\Delta \geq 8$. For $\Delta = 3$, this conjecture has been proved by Thomassen in [154]. In the case of outerplanar graphs with $\Delta = 3$, a stronger result holds as shown by Li and Zhou in [114]. In [115], Lih, Wang and Zhu proved that for any K_4-minor free graph G with $\Delta \geq 4$, $\chi(G^2) \leq (3\Delta/2) + 1$. Since 2-trees are K_4-minor free, this bound holds for them. Combining this with $\omega(G^2) \geq \Delta(G)$, we then have $\omega(G^2) \geq 2(\chi(G^2) - 1)/3$ for any 2-tree G. It turns out that the factor $2/3$ here is the best one can hope for: In Figure 4.9, we give a 2-tree whose square has clique number $2\lambda + 5$ and chromatic number $3\lambda + 3$.

In view of Theorem 4.5, the obvious next step would be to prove Hadwiger's conjecture for squares of k-trees for a fixed $k \geq 3$. Since squares of 2-trees are 2-simplicial graphs, another related problem would be to prove Hadwiger's conjecture for the class of 2-simplicial graphs or some interesting subclasses of it. It is also interesting to work on Hadwiger's conjecture for squares of some other special classes of graphs such as planar graphs.

CHAPTER 5

Rainbow Coloring and its Variants

5.1 Introduction

Graph coloring and graph connectivity are two of the most famous topics in graph algorithms. Many different types of colorings and connectivity measures have been considered throughout time. The concept of rainbow coloring brings these two extensively studied topics together. Rainbow coloring was first defined a decade ago by Chartrand et al. [43] using edge colorings. Let G be a connected, edge-colored graph. A **rainbow path** in G is a path whose edges are all colored with distinct colors. The graph G is said to be *rainbow connected* if there is a rainbow path between every pair of its vertices. The **rainbow connection number** of a graph G, denoted by $\mathsf{rc}(G)$, is the smallest number of colors needed to color the edges of G such that G is rainbow connected under this coloring. Such a coloring is called **Rainbow Coloring (RC)**. The resulting computational problem is as follows.

k-RC
Input: a connected graph G
Output: Is $\mathsf{rc}(G) \leq k$?

This problem has various applications in telecommunications, data transfer, and encryption [110, 31, 53] and has been studied rather thoroughly from both graph-theoretic and complexity-theoretic viewpoints. The rainbow connection number has attracted much attention, and the exact number is known for a variety of simple graph classes [43, 40, 150] and the complexity of computing this number was broadly investigated [8, 17, 31, 40, 41]. See Section 5.1.4 and also the surveys by Li et al. [110, 113] for specifics.

To prove an upper bound on $\mathsf{rc}(G)$, the choice of the path P that is rainbow colored is crucial. The analysis would seem simpler when we are able to choose P as a shortest path between its two endpoints. This leads to the definition of the **strong rainbow connection number** of a graph. Formally, the strong rainbow connection number $\mathsf{src}(G)$ of a graph G is the smallest number of colors needed such that there exists a coloring of $E(G)$ with these colors such that, for every pair of vertices, there exists at least one *shortest* path P between them, such that P is a rainbow path. We call such a coloring as a **Strong Rainbow Coloring (SRC)** of the graph. The resultant computational problem is defined as follows.

k-SRC
Input: a connected graph G
Output: Is $\mathsf{src}(G) \leq k$?

Clearly, $\mathsf{src}(G) \geq \mathsf{rc}(G)$, and both parameters are at least the diameter of G. Moreover, $\mathsf{rc}(G) = 2$ if and only if $\mathsf{src}(G) = 2$ [31]. See section 5.1.4 for further related work on SRC.

5.1.1 Rainbow Coloring and Forest Number

While introducing the rainbow coloring parameters, Chartrand et al. [43] also determined some basic properties and their values for some structured graphs. It is straightforward to verify that $\mathsf{diam}(G) \leq \mathsf{rc}(G) \leq \mathsf{src}(G) \leq m$, where m is the number of edges of G. It is not difficult to see that $\mathsf{rc}(G) = \mathsf{src}(G) = 1$ if and only if G is complete. On the other hand, it holds that $\mathsf{rc}(G) = \mathsf{src}(G) = m$ if and only if G is a tree. Chartrand et al. [43] also determined the exact (strong) rainbow connection numbers for cycle graphs, wheel graphs, and complete multipartite graphs.

It has turned out that domination is a useful concept when deriving upper bounds on $\mathsf{rc}(G)$ (see e.g., [30, 100, 35], see Section 5.1.4 for specifics). In addition to domination, various authors (see e.g., [30]) have noted trees to be useful in bounding $\mathsf{rc}(G)$. More precisely, it is easy to see that $\mathsf{rc}(G) \leq n-1$ by coloring the edges of a spanning tree of G in distinct colors, and assigning an already used color for the remaining edges. Moreover, Kamčev et al. [90] proved that $\mathsf{rc}(G) \leq \mathsf{diam}(G_1) + \mathsf{diam}(G_2) + c$, where $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ are connected spanning subgraphs of G and $c \leq |E_1 \cap E_2|$. Lauri [102, Section 6] suggested bounds on $\mathsf{rc}(G)$ and $\mathsf{src}(G)$ in terms of other, possibly "tree-related", graph parameters. The author used GraPHedron [123], which is a computer-assisted system for automated conjecture-making. In particular, based on exhaustive enumeration on all small graphs (typically with at most eight vertices), the system is able to suggest conjectures in form of inequalities. Two conjectures made by GraPHedron are as follows, and have been verified for all connected simple graphs with at most eight vertices. Notice that in both, $\mathsf{src}(G)$ can be replaced by $\mathsf{rc}(G)$, since $\mathsf{rc}(G) \leq \mathsf{src}(G)$.

Conjecture 5.1 (GraPHedron)**.** *A connected graph G with n vertices and chromatic number $\chi(G)$ has $\mathsf{src}(G) \leq n + 1 - \chi(G)$.*

Conjecture 5.2 (GraPHedron)**.** *A connected graph G with forest number $f(G)$ has $\mathsf{src}(G) \leq f(G) - 1$.*

As stated in [102, Section 6], a weaker form of Conjecture 5.1 is that $\mathsf{src}(G) \leq n + 1 - \omega(G)$, where $\omega(G)$ is the size of a largest clique in G. An even weaker form is $\mathsf{src}(G) \leq n - 1$, which does not follow by the same argument we used for showing that $\mathsf{rc}(G) \leq n - 1$. To the best of our knowledge, the status of these conjectures are open.

To understand Conjecture 5.2, recall that the *forest number* of G, denoted by $f(G)$, is the number of vertices in any maximum induced forest of G. A tightly related parameter is the size of a largest induced tree in G, denoted by $t(G)$, whose study was initiated by Erdös et al. [62]. Clearly, as $t(G) \leq f(G)$, it might be tempting to conjecture that $\mathsf{src}(G) \leq t(G) - 1$. However, this statement is disproved by taking H to be a K_4 with a pendant vertex attached to each of its vertices. In this case, $\mathsf{src}(H) = 4$, but $t(H) = 4$ whereas $f(H) = 6$.

Our Contribution

We make partial progress towards settling Conjecture 5.2. In particular, we prove the following weaker form of it in Section 5.2.

Theorem 5.3. *A connected graph G with forest number $f(G)$ has $\mathsf{rc}(G) \leq f(G) + 2$.*

We also make partial progress towards proving Conjecture 5.1 by showing that $\mathsf{src}(G) \leq n + 1 - \chi(G)$ holds when G is chordal, in Corollary 5.50 of Section 5.3.

5.1.2 Very Strong Rainbow Coloring

Non-trivial upper bounds on $\mathsf{src}(G)$ are known for some simple graph classes such as cycles, wheels, and complete bipartite graphs [43] and block graphs [102]. The recent survey by Li and Sun [113] mentions only one bound on $\mathsf{src}(G)$, which is a bound in terms of the number of edge disjoint triangles of G [111]. The lack of combinatorial bounds on $\mathsf{src}(G)$ for G in specific graph classes is somewhat surprising compared to the vast literature for $\mathsf{rc}(G)$ (see the surveys [110, 112, 113]). Li and Sun [113] explain this by the fact that $\mathsf{src}(G)$ is not a monotone graph property, and thus investigating $\mathsf{src}(G)$ is much harder than investigating $\mathsf{rc}(G)$. Hence, it is a major open question to prove upper bounds on $\mathsf{src}(G)$.

Our Contribution

We make significant progress on the above question. We observe that to prove upper bounds on $\mathsf{src}(G)$, it suffices to prove the existence of a coloring where all edges of not just one, but of *all* shortest paths between two vertices receive different colors. Therefore, we define the **very strong rainbow connection number**, denoted by $\mathsf{vsrc}(G)$, of a graph G, which is the smallest number of colors for which there exists a coloring of $E(G)$ such that, for every pair of vertices and *every shortest* path P between them, all edges of P receive different colors. We call a coloring that achieves this property a **very strong rainbow coloring (VSRC)** of the graph. We also define the corresponding computational problem as below.

k-VSRC
Input: a connected graph G
Output: Is $\mathsf{vsrc}(G) \leq k$?

We prove the first combinatorial upper bounds on $\mathsf{vsrc}(G)$ for several graph classes. These immediately imply upper bounds on $\mathsf{src}(G)$ for the same graph classes. In particular, we show upper bounds that are linear in $|V(G)|$ (improving from the trivial bound of $|E(G)|$) if G is a chordal graph, a circular arc graph, or a disk graph. The upper bound on src that we obtain for chordal graphs proves Conjecture 5.1 for the class of chordal graphs.

Conversely, we prove that a bound on $\mathsf{vsrc}(G)$ implies that G should be highly structured: the neighborhood of every vertex can be partitioned into $\mathsf{vsrc}(G)$ many cliques. For further details, we refer to Section 5.3.

Then, we address the computational complexity of k-VSRC. To start our investigation, we prove the following hardness result on general graphs. See Section 5.4 for the details.

Theorem 5.4. 3-VSRC *is NP-complete. Moreover, there is no polynomial-time algorithm that approximates* $\mathsf{vsrc}(G)$ *within a factor* $|V(G)|^{1-\varepsilon}$ *for any* $\varepsilon > 0$, *unless P=NP.*

This result implies that k-VSRC is not fixed-parameter tractable when parameterized by k, unless P=NP. In order to prove the theorem, we show a nontrivial connection to the clique partition number of a graph.

We remark that, in contrast to the NP-complete 2-Rc and 2-Src problems, 2-VSRC can be solved in polynomial time (see Section 5.6 for the proof). Together with Theorem 5.4, this gives a dichotomy result for the complexity of k-VSRC.

Proposition 5.5. 2-VSRC *can be decided in polynomial time.*

We then study the complexity of determining $\mathsf{vsrc}(G)$ for graphs of bounded treewidth. This is a major open question also for $\mathsf{src}(G)$ and $\mathsf{rc}(G)$ [102], which are only known to be solvable in polynomial time on graphs of treewidth 1. We mention that no results for graphs of higher treewidth are known, even for outerplanar or cactus graphs. However, for the slightly different problem of deciding whether an already given coloring forms a (strong) rainbow coloring of a given graph, a polynomial-time algorithm for cactus graphs and an NP-hardness result for outerplanar graphs are known [155].

With this in mind, we focus on cactus graphs and make the first progress towards understanding the complexity of rainbow coloring problems, in particular of computing $\mathsf{vsrc}(G)$, on graphs of treewidth 2, with the following result.

Theorem 5.6. *Let G be any cactus graph. Then $\mathsf{vsrc}(G)$ can be computed in polynomial time.*

Our algorithm relies on an extensive characterization result for the behavior of very strong rainbow colorings on cactus graphs. Since a cactus graph consists of bridges, even cycles, and odd cycles, we analyze the behavior of any very strong rainbow coloring of the graph with respect to these structures. We show that color repetition can mostly occur only within an odd cycle or even cycle. Odd cycles can repeat some colors from outside but we characterize how they can be repeated. However, our arguments are not sufficient to derive a completely combinatorial bound. Instead, we must find a maximum matching in a well-chosen auxiliary graph to compute the very strong rainbow connection number. See Section 5.5 for further details.

We also observe that $\mathsf{vsrc}(G)$ can be computed efficiently for graphs having bounded treewidth, when $\mathsf{vsrc}(G)$ itself is small. In contrast to known results for the (strong) rainbow connection number [55], we present an algorithm that does not rely on Courcelle's theorem. (See section 5.6 for details.)

Theorem 5.7. *k-VSRC is fixed-parameter tractable when parameterized by $k + \mathsf{tw}$, where tw is the treewidth of the input graph.*

5.1.3 Rainbow Vertex Coloring

The intense interest in Rainbow Coloring led Krivelevich and Yuster [100] to define a natural variant on *vertex-colored* graphs. Here, a path in a vertex-colored graph H is a **rainbow vertex path** if all its *internal* vertices have distinct colors. We say that H is *rainbow vertex connected* if there is a rainbow vertex path between every pair of its vertices. The **rainbow vertex connection number**, denoted by $\mathsf{rvc}(G)$, of a graph G is the smallest number of colors needed to color the vertices of G such that G is rainbow vertex connected under this coloring. Such a coloring is called **Rainbow Vertex Coloring (RVC)**. The resulting computational problem is as follows.

k-RVC
Input: a connected graph G
Output: Is $\mathsf{rvc}(G) \leq k$?

k-RVC is NP-complete for every $k \geq 2$ [48, 47], and remains NP-complete for $k = 3$ for bipartite graphs [108]. In addition, it is NP-hard to approximate $\mathsf{rvc}(G)$ within a factor of $2 - \varepsilon$ unless $\mathsf{P} \neq \mathsf{NP}$, for any $\varepsilon > 0$ [56]. It is also known that k-RVC is linear-time solvable on planar graphs for every fixed k [102]. Finally, assuming the Exponential Time Hypothesis, there is no algorithm for solving k-RVC in time $2^{o(n^{3/2})}$ for any $k \geq 2$ [102].

A stronger variant of rainbow vertex-colorings was introduced by Li et al. [109]. A vertex-colored graph H is *strongly rainbow vertex connected* if between every pair of vertices of H, there is a *shortest* path that is also a rainbow vertex path. A coloring under which G becomes strongly rainbow vertex connected is called a **Strong Rainbow Vertex Coloring (RVC)**. The resulting computational problem is as follows.

k-RVC
Input: a connected graph G
Output: Is $\mathsf{srvc}(G) \leq k$?

This definition is the vertex variant of the k-SRC problem. The **strong rainbow vertex connection number** of G, denoted by $\mathsf{srvc}(G)$, is the minimum k such that G has a strong rainbow vertex coloring with k colors. k-SRVC is NP-complete for every $k \geq 2$ [55] and linear-time solvable on planar graphs for every fixed k [102]. In addition, it is NP-hard to approximate $\mathsf{srvc}(G)$ within a factor of $n^{1/2-\varepsilon}$ unless $\mathsf{P} \neq \mathsf{NP}$, for any $\varepsilon > 0$ [56].

While k-RC has been widely studied in more than 300 published papers, we are unaware of any further complexity results on k-RVC and k-SRVC than those mentioned previously. In particular, the complexity of k-RVC and k-SRVC on structured graph classes is mostly open. This led Lauri [102, Open problem 6.6] to ask the following:

For what restricted graph classes do k-RVC *and* k-SRVC *remain NP-complete?*

Our Contribution

We make significant progress towards addressing the above question. In particular, we study bipartite graphs and chordal graphs, and some of their subclasses, and give hardness results and polynomial-time algorithms for k-RVC and k-SRVC (See Sections 5.7 and 5.8). Our main result is a hardness result for bipartite apex graphs:

Theorem 5.8. *Let G be a bipartite apex graph of diameter 4. It is NP-complete to decide both whether $\mathsf{rvc}(G) \leq 4$ and whether $\mathsf{srvc}(G) \leq 4$. Moreover, it is NP-hard to approximate $\mathsf{rvc}(G)$ and $\mathsf{srvc}(G)$ within a factor of $5/4 - \varepsilon$, for every $\varepsilon > 0$.*

This result is particularly interesting since no hardness result was known on a sparse graph class (like apex graphs) for any of the variants of rainbow coloring. Moreover, this result can be considered tight in conjunction with the known result that k-RVC and k-SRVC are linear-time solvable on planar graphs for every fixed number of colors k [102]. Finally, we observe (like Li et al. [108]) that $\mathsf{rvc}(G)$ and $\mathsf{srvc}(G)$ can be computed in linear time if G is a bipartite graph of diameter 3, providing further evidence that this result is tight.

For general bipartite graphs and for split graphs (a well-known subclass of chordal graphs), we exhibit stronger hardness results:

Theorem 5.9. *Let G be a bipartite graph of diameter 4. It is NP-complete to decide both whether $\mathsf{rvc}(G) \leq k$ and whether $\mathsf{srvc}(G) \leq k$, for every $k \geq 3$. Moreover, it is NP-hard to approximate both $\mathsf{rvc}(G)$ and $\mathsf{srvc}(G)$ within a factor of $n^{1/3-\varepsilon}$, for every $\varepsilon > 0$.*

We remark that, previously, it was only known that deciding whether $\mathsf{rvc}(G) \leq 3$ for bipartite graphs G is NP-complete by the result of [108]. Our construction, however, is conceptually simpler, gives hardness for every $k \geq 3$, and is easily extended to the strong variant. Moreover, for k-RVC on general graphs, this result implies a considerable improvement over the previous result of Eiben et al. [56] which only excluded a polynomial-time approximation with a factor of less than 2 assuming $\mathsf{P} \neq \mathsf{NP}$.

Theorem 5.10. *Let G be a split graph of diameter 3. It is NP-complete to decide both whether $\mathsf{rvc}(G) \leq k$ and whether $\mathsf{srvc}(G) \leq k$, for every $k \geq 2$. Moreover, it is NP-hard to approximate both $\mathsf{rvc}(G)$ and $\mathsf{srvc}(G)$ within a factor of $n^{1/3-\varepsilon}$, for every $\varepsilon > 0$.*

To the best of our knowledge, our results for split graphs give the first non-trivial graph class besides diameter-2 graphs for which the complexity of the edge and the vertex variant differ (see e.g, [102, Table 4.2] but note that it contains a typo erroneously claiming that k-RVC can be solved in polynomial-time for split graphs). In particular, k-RC can be solved in polynomial time on split graphs when $k \geq 4$ [40, 42]. Moreover, we observe that $\mathsf{rvc}(G)$ and $\mathsf{srvc}(G)$ can be computed in linear time if G is a graph of diameter 2, providing evidence that this result is tight.

To contrast our hardness results, we show that both problems can be solved in polynomial time on several other subclasses of bipartite graphs and chordal graphs.

Theorem 5.11. *If G is a bipartite permutation graph, a block graph, or a unit interval graph, then $\mathsf{rvc}(G)$ and $\mathsf{srvc}(G)$ can be computed in linear time. If G is an interval graph, then $\mathsf{rvc}(G)$ can be computed in linear time.*

Combined, these results paint a much clearer picture of the complexity landscape of k-RVC and k-SRVC than was possible previously.

5.1.4 Further Related Work

A basic introduction to rainbow coloring can be found in Chapter 11 of the book Chromatic Graph Theory by Chartrand and Zhang [43] and we refer to [102, 110] for detailed surveys on rainbow coloring and its variants. The concept of rainbow coloring was introduced by Chartrand, Johns, McKeon, and Zhang [43] in 2008. The rainbow connection number and strong rainbow connection number have been studied from both algorithmic and graph-theoretic points of view. The exact rainbow connection numbers are known for a variety of simple graph classes, such as wheel graphs, complete multipartite graphs [43], unit interval graphs [150], and threshold graphs [40].

Much of the research on rainbow connectivity has focused on finding bounds on the parameters, either in terms of the number of vertices n or some other well-known parameters. For 2-connected graphs, Ekstein et al. [57] showed that $\mathsf{rc}(G) \leq \lceil n/2 \rceil$, and this is tight as witnessed by e.g., odd cycles. Further, it has turned out that domination is a useful concept when deriving upper bounds on $\mathsf{rc}(G)$ (see e.g., [30, 100, 35]). Specifically, Chandran et al. [35] proved that $\mathsf{rc}(G) \leq \frac{3n}{\delta+1} + 3$, where δ denotes the minimum degree of G. Moreover, the authors derived that when $\delta \geq 2$, then $\mathsf{rc}(G) \leq \gamma_c(G) + 2$, where $\gamma_c(G)$ is the connected domination number (see preliminaries in Section 5.1.5). For some structured graph classes, this leads to upper bounds of the form $\mathsf{rc}(G) \leq \mathsf{diam}(G) + c$, where c is a small constant. For instance, it follows that $\mathsf{rc}(G) \leq \mathsf{diam}(G) + 1$ where G is an interval graph with $\delta \geq 2$, and $\mathsf{rc}(G) \leq \mathsf{diam}(G) + 3$, where G is an AT-free graph with $\delta \geq 2$. Note that these bounds are almost tight as $\mathsf{diam}(G)$ is a lower bound for $\mathsf{rc}(G)$. For a more comprehensive treatment, we refer the curious reader to the books [44, 107] and the surveys [110, 113].

Rainbow coloring is a notoriously hard problem computationally. It was shown by Chakraborty et al. [31] that 2-RC is NP-complete, and Ananth et al. [8] showed that for any $k > 2$, k-RC is NP-complete. Later, Chandran and Rajendraprasad [40] strengthened this result to hold even for chordal graphs. Basavaraju et al. [17] gave an $(r + 3)$-factor approximation algorithm to rainbow color a graph of radius r. Chandran and Rajendraprasad [41] proved that there is no polynomial time algorithm that approximates rainbow connection number of a graph within a factor smaller than 2. On split graphs, k-RC is NP-complete when $k \in \{2, 3\}$, but solvable in polynomial time otherwise [40, 42]. It is also solvable in polynomial time on threshold graphs [40]. On bridgeless chordal graphs, there is a linear-time (3/2)-approximation algorithm for k-RC, however the problem cannot be approximated with a factor less than 5/4 on this graph class, unless $\mathsf{P} = \mathsf{NP}$ [41]. Recently, Kowalik et al. showed that for any $k \geq 2$, k-RC cannot be solved in $2^{o(n^{3/2})}$ or $2^{o(m/\log m)}$ time, where $n = |V(G)|$ and $m = |E(G)|$, unless ETH fails [97]. Agrawal improved the lower bound to $2^{o(m)}$ [4].

It was shown in [31] that $rc = 2$ if and only if $src = 2$ and hence k-SRC is NP-complete for $k = 2$. The problem remains NP-complete for bipartite graphs [8] and split graphs [103]. In [8], it is shown that the strong rainbow connection number of a bipartite graph of n vertices cannot be approximated within a factor of $n^{1/2-\varepsilon}$, where $\varepsilon > 0$ unless $NP = ZPP$. The same holds for split graphs [103].

For $k = 2$, it is not difficult to verify that k-RC is equivalent to k-SRC. Not surprisingly, k-SRC is also NP-complete for $k \geq 2$ [8]. In contrast to k-RC, k-SRC remains hard on split graphs for every $k \geq 2$ [102, Theorem 4.1]. Moreover, on n-vertex split graphs, it is NP-hard to approximate k-SRC within a factor of $n^{1/2-\varepsilon}$ for any $\varepsilon > 0$, while k-RC admits an additive-1 approximation [40]. The former statement also holds for n-vertex bipartite graphs instead of split graphs [8]. For block graphs, computing k-SRC can be done in linear time [93], while k-RC on block graphs is conjectured to be hard (see [102, Conjecture 6.3] or [93]). In general, it appears that despite the interest, there are fewer complexity-theoretic results on k-SRC. In fact, the same is true when considering combinatorial results (see [110] for a broader discussion).

Some other variants of rainbow problems have been studied as well. When a coloring of the edges or the vertices of a graph is already given as input, we can ask whether the graph is rainbow-connected or rainbow vertex-connected. Both of these problems are known to be NP-complete even on highly restricted graphs, like interval graphs, series-parallel graphs, and k-regular graphs for every $k \geq 3$ [104, 103, 155]. However, we stress that these problems are strictly different from k-RC and k-RVC. That is, complexity results on one problem are not transferable to the other.

5.1.5 Preliminaries

Given a digraph G, we denote by $\tilde{G}$, the underlying undirected (multi) graph of it, by which we mean that the vertex set is the same and for every directed edge $\overrightarrow{uv}$ in G, we put an undirected edge between u and v in $\tilde{G}$. For any two vertices u, v in a tree T, let T_{uv} denote the unique path between u and v in T.

A *block* of a graph is a maximal 2-connected component. A *cactus graph* is a graph in which each block of the graph is a cycle or an edge; equivalently, a cactus graph is a graph in which every edge belongs to at most one cycle.

For a graph G, let $\hat{G}$ denote the graph obtained by adding a new vertex $\hat{u}$ to G such that $\hat{u}$ is adjacent to all vertices of G, i.e., $\hat{u}$ is a *universal vertex* in $\hat{G}$. We use $\mathsf{cp}(G)$ to denote the *clique partition number* (or clique cover number) of G, the smallest number of subsets of $V(G)$ that each induce a clique in G and whose union is $V(G)$.

We use $\mathsf{is}(G)$ to denote the smallest size of the universe in any intersection graph representation of G, and $\mathsf{ecc}(G)$ to denote the smallest number of cliques needed to cover all edges of G. It is known that $\mathsf{is}(G) = \mathsf{ecc}(G)$ [139].

We define some natural generalizations of line graphs. A graph is *k-perfectly groupable* if the neighborhood of each vertex can be partitioned into k or fewer cliques. It is well known that line graphs are 2-perfectly groupable. A graph is *k-perfectly orientable* if there exists an orientation of its edges such that the outgoing neighbors of each vertex can be partitioned into k or fewer cliques. Clearly, any k-perfectly groupable graph is also k-perfectly orientable. Many geometric intersection graphs, such as disk graphs, are known to be k-perfectly orientable for small k [91].

A *d-distance coloring* of G is a coloring c of G such that $c(u) \neq c(v)$ whenever $\mathsf{dist}(u, v) \leq d$. The minimum number of colors needed for a d-distance coloring of G is known as the *d-distance chromatic number* of G, and it is denoted by $\chi_d(G)$. Note that $\chi_d(G)$ is equivalent to $\chi(G^d)$, i.e., the chromatic number of the d^{th} power of G.

The parameter strong rainbow vertex coloring ($\mathsf{srvc}(G)$) was defined by Li et al. [109], and they also verified that $\mathsf{diam}(G) - 1 \leq \mathsf{rvc}(G) \leq \mathsf{srvc}(G) \leq |V(G)| - 2$. The following upper bound was mentioned in [102] (see the same reference for further discussion and examples).

Proposition 5.12 ([102]). *Let G be a connected graph with $\mathsf{diam}(G) = d \geq 3$. Then*

$$d - 1 \leq \mathsf{rvc}(G) \leq \mathsf{srvc}(G) \leq \chi_{d-2}(G).$$

Proof. There are at least two vertices in G connected by a shortest path of length d. Clearly, every coloring must use at least $d - 1$ colors to rainbow-connect this pair. On the other hand, between every pair of vertices u and v, there is a path of length at most d, meaning that it contains at most $d - 1$ internal vertices. As every $(d - 2)$-distance coloring colors these internal vertices distinctly, the statement follows. $\square$

A *dominating set* of G is a set $D \subseteq V$ such that every vertex in $V \setminus D$ is adjacent to at least one vertex in D. If $G[D]$ is connected, then D is a *connected dominating set*. The minimum size of a connected dominating set in G, denoted by $\gamma_c(G)$, is known as the *connected domination number* of G. This parameter provides an upper bound on the rainbow vertex connection number of a connected graph, since G becomes rainbow vertex-connected by simply coloring all vertices of the connected dominating set distinctly, and the remaining vertices with any of the already used colors. This observation can be derived from [100].

Proposition 5.13 ([100])**.** *If G is a connected graph, then* $\mathsf{rvc}(G) \leq \gamma_c(G)$.

We will use the following notions about bipartite graphs: A *strong ordering* $<$ of the vertices of a bipartite graph G consists of an ordering of $L(G)$ and an ordering of $R(G)$ such that for all pair of edges $uv, u'v' \in E$, where $u, u' \in L(G)$ and $v, v' \in R(G)$, if $u < u'$ and $v' < v$, then $uv', u'v \in E$. A bipartite graph G is a *chain graph* if the vertices of $L(G)$ can be ordered as $\{a_1, a_2, \ldots, a_k\}$ such that $N(a_1) \subseteq N(a_2) \subseteq \cdots \subseteq N(a_k)$. Note that if G is a chain graph then $R(G)$ also has an ordering $\{b_1, b_2, \ldots b_\ell\}$ such that $N(b_1) \subseteq N(b_2) \subseteq \cdots \subseteq N(b_\ell)$.

Hard Problems

For our hardness reductions we will use some well-known NP-complete problems. The first among them is called Hypergraph k-Coloring. A *hypergraph* $H = (N, \mathcal{E})$ with vertex set N and hyperedge set $\mathcal{E}$ is a generalization of a graph, in which edges can contain more than two vertices. Thus $\mathcal{E}$ consists of subsets of N of arbitrary size. The definition of a (vertex) coloring of a hypergraph is a generalization of the vertex coloring of a graph. In a colored hypergraph, an edge is called *monochromatic* if all of its vertices received the same color. A *proper coloring* of a hypergraph generalizes a proper coloring of a graph in a natural way: we require that no hyperedge is monochromatic. To avoid trivial cases, we can assume from now on that every hyperedge contains at least two vertices. Thus a proper coloring must always use at least two colors.

Hypergraph k-Coloring
Input: a hypergraph H
Output: Is there a proper coloring of H with at most k colors

Hypergraph k-Coloring is well-known to be NP-complete for every $k \geq 2$ [117]. We repeat the definition of the problem Coloring for easy reference.

Coloring
Input: An undirected graph G
Output: The smallest k such that G has a proper k-coloring
.

Coloring is NP-hard to approximate within a factor of $n^{1-\varepsilon}$ for any $\varepsilon > 0$, where n is the number of vertices [164]. The following problem is known to be NP-complete [71].

Planar 3-Coloring
Input: a planar graph G
Output: Does G have a proper 3-coloring?

5.2 RC and Forest Number

Let G be a connected graph, $V = V(G)$ and $E = E(G)$. Let rc denote the rainbow connection number of G and f denote the forest number of G. We will prove that $rc \leq f + 2$, thereby proving Theorem 5.3.

5.2.1 Take 1: $\mathsf{rc} \leq 3f - 1$

In this section, we prove a much easier weaker bound, which is that $\mathsf{rc} \leq 3f - 1$.

Lemma 5.14. *If there is a set $D \subseteq V$ such that $G[D]$ is connected and every vertex in $V \setminus D$ has at least 2 neighbors in D, then $\mathsf{rc} \leq |D| + 1$.*

Proof. Since $G[D]$ is connected, it has at least one spanning tree. Pick an arbitrary spanning tree τ of $G[D]$ and color its edges with distinct colors from 1 to $|D| - 1$. Clearly, $G[D]$ is now rainbow-connected. We now extend the rainbow-connection to rest of the graph. Every vertex $v \in V \setminus D$ has at least 2 neighbors in D by the second precondition of the lemma. For each $v \in V \setminus D$, pick any 2 neighbors in D; call them $d_1(v)$ and $d_2(v)$. Color the edge $vd_1(v)$ with color $|D|$ and $vd_2(v)$ with color $|D| + 1$. Now let us prove that there is a rainbow path between any $u, v \in V(G)$.
Case 1. $u, v \in V(D)$: There is a rainbow path between u and v as $G[D]$ is rainbow-connected via the spanning tree τ.
Case 2. $u \in V \setminus D$ and $v \in D$: There is a rainbow path between $d_1(u)$ and v via τ and this path uses only colors from 1 to $|D| - 1$. Since $ud_1(u)$ is colored with color $|D|$, we have a rainbow path between u and v.
Case 3. $u \in D$ and $v \in V \setminus D$: Similar to case 2.
Case 4. $u, v \in V \setminus D$: There is a rainbow path between $d_1(u)$ and $d_2(v)$ via τ and this path uses only colors from 1 to $|D| - 1$. Since $ud_1(u)$ is colored with color $|D|$ and $vd_2(v)$ is colored with color $|D| + 1$, we have a rainbow path between u and v. □

Let $\mathcal{F}$ be a maximum induced forest of G. Let F be the set of vertices in $\mathcal{F}$. Let $\mathcal{T}$ denote the set of connected components (trees) of $\mathcal{F}$.

Lemma 5.15. *For any maximum induced forest $\mathcal{F}$ of G and for any $v \in V \setminus V(\mathcal{F})$, there exists a connected component (tree) of $\mathcal{F}$ such that, v has at least 2 neighbors in T.*

Proof. Otherwise, $G[F \cup \{v\}]$ is a forest, which contradicts the maximality of $\mathcal{F}$. □

Lemma 5.16. *There exists a vertex set $A \subseteq V \setminus F$ of size at most $2|F| - 2$ such that $G[F \cup A]$ is connected.*

Proof. Let B be the smallest subset of $V \setminus F$ such that $G[F \cup B]$ has a smaller number of connected components than $G[F]$. We show that $|B| \leq 2$. From the minimality of B, it follows that B induces a path in G that connects two trees in $\mathcal{T}$. Let $b_1, b_2, \cdots, b_k$ be the vertices in this path, where b_1 and b_k are adjacent to distinct trees T_1 and T_2 respectively. Due to Lemma 5.15, b_2 has an edge to some tree $T \in \mathcal{T}$. If $T = T_1$, then $B \setminus \{b_1\}$ would also connect T_1 and T_2. Hence $T \neq T_1$. But then the vertex set $\{b_1, b_2\}$ is sufficient to connect T and T_1 and thereby reduce the number of components. This implies $k = 2$ and hence $|B| \leq 2$. Now, we add B to F to reduce the number of connected components by at least 1. Then we can repeat the process again until we get a single connected component. At each stage, the currently existing connected components take the role of the T_i's. We only need to repeat the process at most $|F| - 1$ times until we get a single connected component. Since each repetition adds at most 2 vertices, the total number of vertices that we add is at most $2|F| - 2$. $\square$

Lemmas 5.14, 5.15 and Lemma 5.16 together imply that $\mathsf{rc} \leq 3f - 1$ as follows. By Lemma 5.16, we have a set A such that $G[F \cup A]$ is connected and $|A| \leq 2|F| - 2$. Let $D = F \cup A$. By Lemma 5.15, we have that each vertex in $V \setminus D$ has at least 2 neighbors in D. Thus D satisfies both preconditions of Lemma 5.14 and hence

$$\mathsf{rc} \leq |D| + 1 \leq |F| + |A| + 1 \leq |F| + 2|F| - 2 + 1 = 3f - 1.$$

5.2.2 Take 2: $\mathsf{rc} \leq 2f + 2$

In this section we strengthen the bound to $2f + 2$. Many of the concepts and techniques that we use in the final proof are already introduced here. See Figure 5.1 for an illustration of this section.

Let $\mathcal{F}$ be a maximum induced forest of G. Let F be the set of vertices in $\mathcal{F}$. Let $\mathcal{T}$ be the set of trees (connected components) of $\mathcal{F}$ and $t = |\mathcal{T}|$ be the number of trees. Let $S := V \setminus F$. We call an edge uv of G a **tree-edge** if both u and v belong to the same tree in $\mathcal{T}$; otherwise the edge is called a **non-tree edge**.

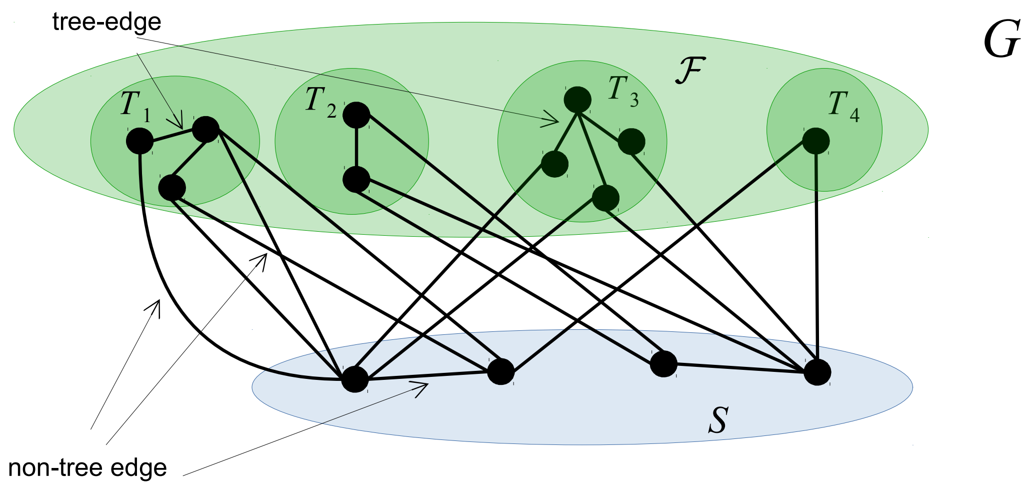
Let H be the graph obtained from G by contracting each tree in $\mathcal{T}$ to a single vertex. Formally, we define H as follows:

$$\begin{aligned} V(H) &:= V_T \cup S \text{ where,} \\ V_T &:= \{x_T : T \in \mathcal{T}\} \\ E(H) &:= E(G[S]) \cup \{ux_T : u \in S,\ T \in \mathcal{T},\ u \text{ has at least 1 edge to } V(T) \text{ in } G\} \end{aligned}$$

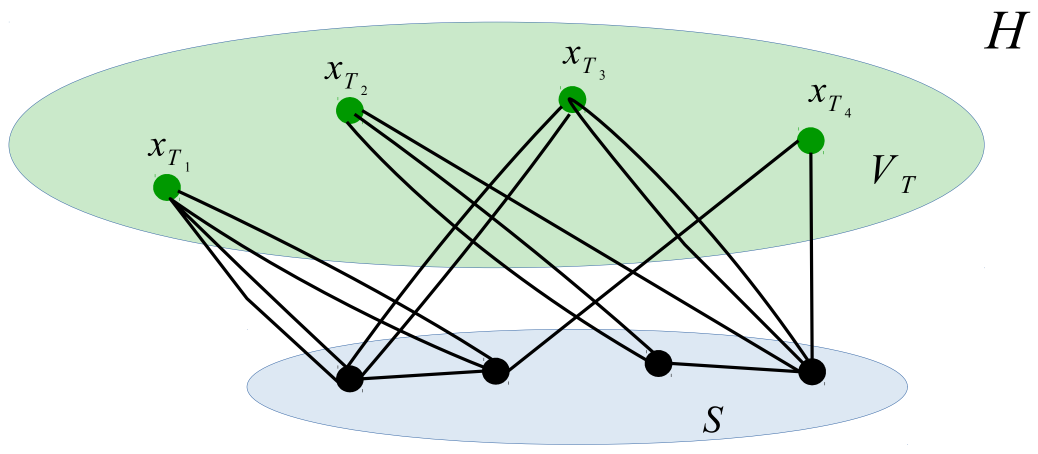
We call the vertices in V_T, the **tree vertices** and the vertices in S, the **non-tree vertices** of H. Notice that V_T is an independent set in H, because there are no edges in G between any two distinct connected components of $\mathcal{F}$. We divide the edges of H into two groups as follows:

$$\begin{aligned} E_1 &:= E(G[S]) \cup \{ux_T : u \in S,\ T \in \mathcal{T},\ u \text{ has exactly 1 edge to } V(T) \text{ in } G\} \\ E_2 &:= \{ux_T : u \in S,\ T \in \mathcal{T},\ u \text{ has at least 2 edges to } V(T) \text{ in } G\} \end{aligned}$$

We call the edges in E_1 as **1-edges** and those in E_2 as **2-edges.** We define a function $f_T : V_T \to \mathcal{T}$ that maps a tree vertex to its corresponding tree, i.e., $f_T(x_T) = T$. For each edge in H, we define its **representatives** in G as follows. Consider first a 2-edge e between $u \in S$ and $x_T \in V_T$. By definition of a 2-edge, u has at least 2 edges to $V(T)$ in G. We arbitrarily choose 2 of these edges as the representatives in G of the 2-edge e and denote them by $(e)_1$ and $(e)_2$. For a 1-edge e between $u \in S$ and $x_T \in V_T$, there is a unique edge between u and $V(T)$ in G, by the definition of a 1-edge. We call this edge, the representative of ux_T in G, denoted by $(e)_1$. For a 1-edge e between $u \in S$ and $v \in S$, we call uv itself as the representative in G. For a 2-edge ux_T with its representatives in G as uv_1 and uv_2, we call the vertices v_1 and v_2, the **foots** of ux_T in T.

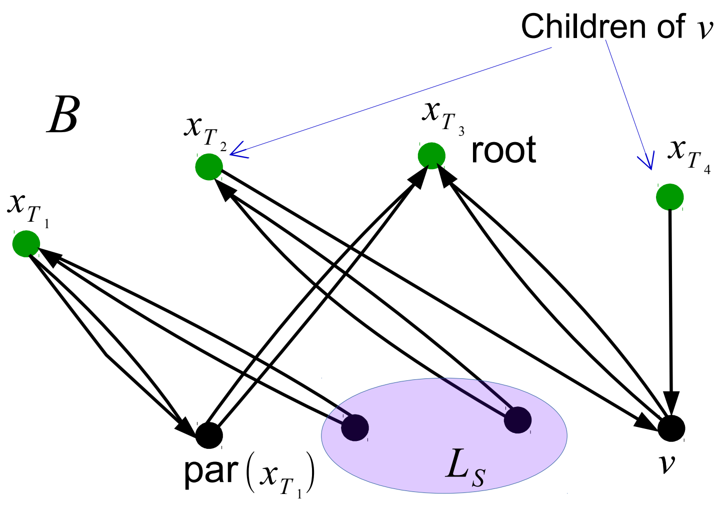


(a) A maximum induced forest $\mathcal{F}$ of graph G

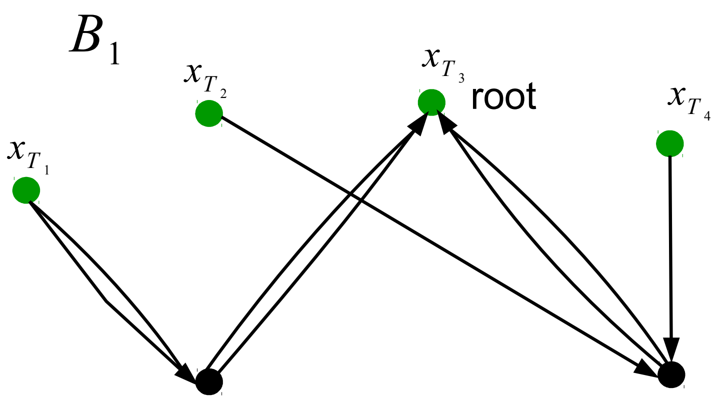


(b) The graph H obtained by contraction of trees in $\mathcal{F}$. We use two lines to draw a 2-edge and one line to draw a 1-edge.

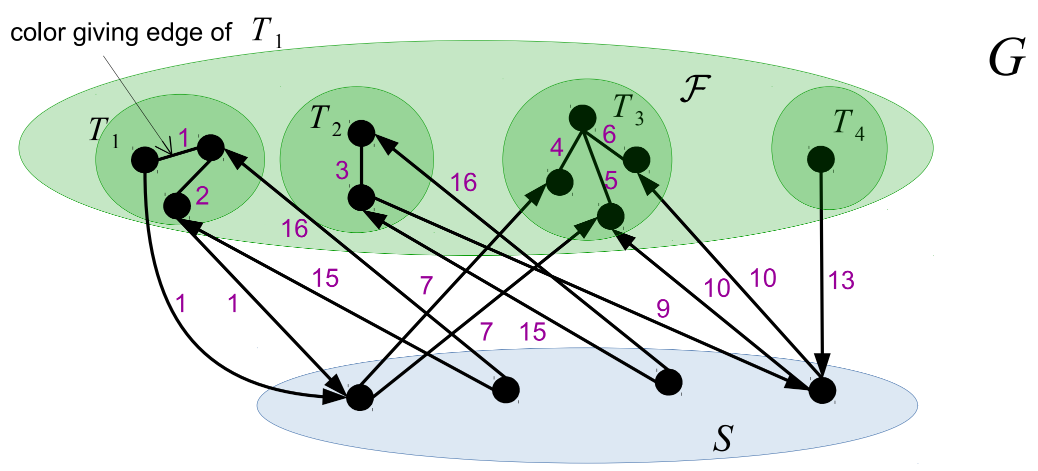
Figure 5.1: A graph G, a maximum induced forest $\mathcal{F}$ of it, the contracted graph H, the skeleton B, the directed tree B_1, and their coloring according to Take 2. The figure is continued on the next 2 pages.



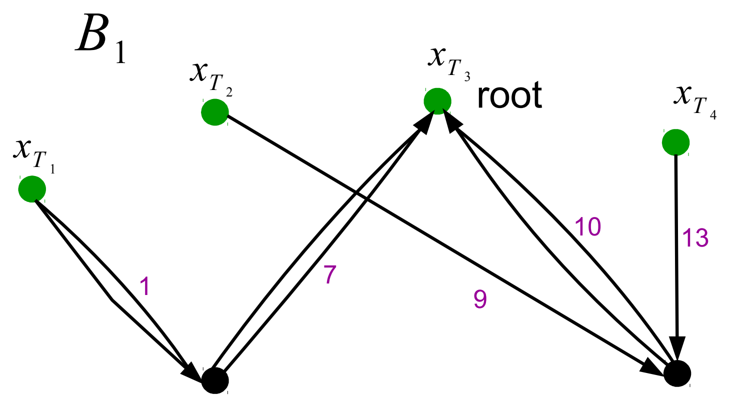
(c) The skeleton B



(d) The directed tree B_1



(e) Our rainbow coloring in Take 2 of the maximum induced forest of graph G. Only the *relevant edges* of G are shown in the figure. The direction of edges across S and $\mathcal{F}$ shown are according to the direction in B_1. Note that the surplus colors are $s_1(T_1) = 7, s_2(T_1) = 8, s_1(T_2) = 9, s_2(T_2) = 10, s_1(T_3) = 11, s_2(T_3) = 12, s_1(T_4) = 13$, $s_2(T_4) = 14$ and the global surplus colors are $g_1 = 15$ and $g_2 = 16$.



(f) Coloring of B_1 according to the coloring procedure in Take 2.

A **skeleton** is defined as a spanning tree of H with one of the nodes in V_T fixed as its root and all the edges of it directed towards the root. Let B be a skeleton such that the number of 2-edges in B is as large as possible (or equivalently the number of 1-edges is as small as possible, as the total number of edges in a skeleton is always $|V(G)| - 1$). The *parent* of a non-root vertex v in B, denoted by $\mathsf{par}(v)$, is the other endpoint of the unique outgoing edge from v. The *children* of v are the other endpoints of the incoming edges of v. Let L_S be the set of vertices of S that are leaves of B and let B_1 be the directed sub-tree of B defined as $B_1 := B[V(B) \setminus L_S]$. For a tree T whose outgoing edge is a 2-edge, we define the **color giving edge** of T as follows: Let e be the outgoing 2-edge; v_1 and v_2 be the foots of e in T. We allocate an arbitrary edge in $T_{v_1 v_2}$ as its color-giving edge.

Lemma 5.17. *Every vertex in S has at least one 2-edge incident on it in B.*

Proof. Suppose v is a vertex in S that has only 1-edges incident on it in B. By Lemma 5.15, there exist a $T \in \mathcal{T}$ such that v has at least 2 edges to T in G. Let C be the connected component of $B \setminus v$ that contains the vertex x_T. Let e be the unique edge in B between v and C. Removing e from B and adding 2-edge vx_T gives a skeleton with higher number of 2-edges than B. This is a contradiction to the choice of B. □

The above lemma has the following corollaries.

Corollary 5.18. *For every vertex in L_S, the unique edge incident on it in B is a 2-edge.*

Corollary 5.19. *Every leaf of B_1 is a tree vertex.*

Proof. Suppose for the sake of contradiction that there is a leaf v of B_1 that is a non-tree vertex. Clearly, $v \notin L_S$ by the definition of B_1. Hence v is not a leaf of B. Thus there must be a vertex u in L_S that has an edge to v in B. Since both $u, v \in S$, the edge uv is a 1-edge. This is a contradiction to Corollary 5.18. □

Coloring procedure: We now give a rainbow coloring of G using $f + t + 2 \leq 2f + 2$ colors. Since $\mathcal{F}$ is a forest with t connected components, the number of edges in $\mathcal{F}$ is $f - t$. Color all the edges in $\mathcal{F}$ with distinct colors $1, \ldots, f - t$. We call colors $g_1 := f + t + 1$ and $g_2 := f + t + 2$ the **global surplus colors**. We use the global surplus colors to color the representatives of those edges of B that are incident on the vertices in L_S. Each vertex in L_S has only one edge incident on it in B, and this edge is a 2-edge due to Corollary 5.18. Color the two representatives of this 2-edge, one with g_1 and the other with g_2. Now there are $2t$ colors, i.e. colors $f - t + 1$ to $f + t$ that are unused so far. We allocate 2 each of these colors as the **surplus colors** of the trees in $\mathcal{T}$. That is, the i^{th} tree in $\mathcal{T}$ is allocated colors $f - t + 2(i - 1) + 1$ and $f - t + 2(i - 1) + 2$ as its surplus colors. We denote the two surplus colors of T by $s_1(T)$ and $s_2(T)$.

Finally, we give a coloring of the edges of B_1 and then extend the coloring to their representatives in G. Whenever we color an edge e in B_1 with color c, we also color the representatives of e in G also with c, though we may not mention this explicitly. Pick each $T \in \mathcal{T}$ such that x_T is not the root of B and do the following: Let $\overrightarrow{x_T v}$ be the outgoing edge of vertex x_T in B, i.e., $v = \mathsf{par}(x_T)$; let $w = \mathsf{par}(v)$ (if parent of v exists, i.e., if v is not the root) and let $z = \mathsf{par}(w)$ (if w exists and parent of w exists, i.e., if w is not the root).

Case 1: If $x_T v$ is a 2-edge: If $x_T v$ is uncolored, then we color it with the same color as that of the color-giving edge of T. Note that the color-giving edge is already colored as we have colored all edges inside F. If w exists and vw is uncolored, color vw with $s_1(T)$, the first surplus color of T. If z exists and wz is uncolored, color wz with $s_2(T)$, the second surplus color of T.
Case 2: If $x_T v$ is a 1-edge: If $x_T v$ is uncolored, then color $x_T v$ with $s_1(T)$. If w exists and vw is uncolored, then color vw with $s_2(T)$.
We prove in Lemma 5.20 that the above procedure in fact colors all the edges of B_1. We extend this coloring of edges of B_1 to their representatives in G as mentioned before: for each edge e of B_1 color their representatives (both representatives in case of 2-edges and the only representative in case of 1-edge) with the color of e.

There might be some edges in G that are still uncolored. We call them *irrelevant edges*. They are called irrelevant because when we exhibit rainbow paths between pairs of vertices later, these edges will not be used. We call the edges of G that are not irrelevant, *relevant edges*. Color all the irrelevant edges with any arbitrary color from $[f + t + 2]$, just to complete the coloring.

Lemma 5.20. *All the edges of B_1 are colored and they are colored with distinct colors.*

Proof. It is easy to see that the colors are distinct as each color is used only once while coloring B_1. It only remains to prove that all edges are colored. Consider any directed edge $\overrightarrow{uv}$ of B_1. If u is a tree vertex i.e, if $u \in \mathcal{T}$, then it is easy to see that $\overrightarrow{uv}$ is colored by the procedure. Hence assume that u is not a tree vertex. Since $u \notin V_T$, u is not a leaf of B_1 by Corollary 5.19, and hence has at least one child in B_1. Let x be one such child. If $x \in V_T$, then it is also easy to see that $\overrightarrow{uv}$ is colored by the procedure. Hence assume that $x \notin V_T$. Now, the edge xu is a 1-edge (as both endpoints are in S). Then, by Lemma 5.21, a child x' of x is a tree vertex with xx' being a 2-edge. Then due to Case 1 of the coloring procedure, edge $\overrightarrow{uv}$ gets colored. □

Lemma 5.21. *For each 1-edge $\overrightarrow{uv}$ in B, either u is a tree vertex, or a child u' of u is a tree vertex with $u'u$ being a 2-edge.*

Proof. Suppose u is not a tree vertex. Then there is an incoming 2-edge on u, because its outgoing edge is a 1-edge and there has to be at least one 2-edge incident on it due to Lemma 5.17. Let the other endpoint of this edge be u'. Since at least one of the endpoints of a 2-edge has to be a tree vertex, u' is a tree vertex. □

Observation 5.22. *Consider a tree T and let v_1, v_2, and v_3 be any 3 vertices in T. Let e be an edge in $T_{v_2 v_3}$. Then, either $T_{v_1 v_2}$ or $T_{v_1 v_3}$ does not contain the edge e.*

We are now ready to prove that the above coloring of the edges of G is indeed a rainbow coloring. We will prove that there is a rainbow path between every pair of vertices in G. For this, first we prove in Lemma 5.23 that there is a rainbow path between any pair of vertices in $V(G) \setminus L_S$ and then in Lemma 5.24, we show that there is a rainbow path between any pair of vertices in $V(G)$.

For a vertex v in $V(G)$, if $v \in S$ define $h(v) := v$, otherwise (i.e., if $v \in F$) define $h(v) := x_T$, where $T \in \mathcal{T}$ is the tree containing v.

Lemma 5.23. *For any pair of vertices $v_1, v_2 \in V(G) \setminus L_S$, there is a rainbow path between v_1 and v_2 in G that uses only colors in $[f+t]$.*

Proof. Now, consider a pair of vertices $v_1, v_2 \in V(G) \setminus L_S$. We will construct a rainbow path P from v_1 to v_2 using only edges of G having colors from $[f+t]$. From Lemma 5.20, we know that there is a rainbow path between $v'_1 := h(v_1)$ and $v'_2 := h(v_2)$ in B_1 whose edges are completely contained in B_1. Let this path be P'. We will use the path P' as a guide to construct our required rainbow path P in G. First break P' into two paths P'_1 and P'_2 as follows: let v'_3 be the least common ancestor of v'_1 and v'_2 in B_1; let P'_1 be the path from v'_1 to v'_3 and P'_2 be the path from v'_2 to v'_3.

We will first construct a path P_1 in G starting from v_1 using the path P'_1 as a guide: we start with P_1 being just the vertex v_1. We maintain a current vertex in G, denoted by v_G, which is initialized to v_1.

Now, as long as $h(v_G) \neq v'_3$, repeat the following step:
Let e be the outgoing edge from $h(v_G)$ in B_1. Note that e is the next edge in P'_1 that we have not processed yet.
Case 1. If $h(v_G) \in S$: Append $(e)_1$ to P_1. Also, update v_G to be the other endpoint of $(e)_1$.
Case 2. If $h(v_G) \in V_T$: Let $x_T = h(v_G)$.
Case 2.1. If e is a 1-edge: Let z be the endpoint of e in T. Append the path $T_{v_G z}$ to P_1. Now, append $(e)_1$ to P_1. Update v_G to be the endpoint of $(e)_1$ outside T.
Case 2.2. If e is a 2-edge: Let z_1 and z_2 be the foots of e in T. Observe that $v_G \in V(T)$ as $h(v_G) = x_T$. By Observation 5.22, there exist a $z \in \{z_1, z_2\}$ such that there is a path from v_G to z that excludes the color-giving edge of T. Append this path to P_1. Now, append to P_1 the representative of e having one of the endpoints as z. Update v_G to be the endpoint outside T of this representative.

Similarly we also construct P_2 starting from v_2 using P'_2 as a guide.

If $v'_3 \in S$, then we take $P := P_1P_2$. Otherwise if $v'_3 \in V_T$ we define P as follows: Let $T = f_T(v'_3)$. There are vertices $w_1, w_2 \in V(T)$ that are the ending vertices of P_1 and P_2 respectively. Let P_3 be the path $T_{w_1w_2}$. Take $P := P_1P_3P_2$.

By construction, it is clear that P is indeed a path from v_1 to v_2. It only remains to prove that P is a rainbow path. First note that we have only used relevant edges with colors from $[f+t]$ in P. In our coloring of the edges of G, each of the colors $1, 2 \cdots, f-t$ except the colors of the color-giving edges, have been used only for one relevant edge. Recall that the color of the color giving edge of a tree T can possibly be repeated on the representatives of the outgoing edge of x_T in B. But when constructing P, we have taken care not to include the color-giving edge of T in P if P contains a representative of the outgoing edge in of x_T. Also, we have included at most one of the representatives of the outgoing edge of x_T in P. Thus the colors $1, 2, \cdots, f-t$ appear at most once in P. Each of the colors from $f-t+1$ to $f+t$ (surplus colors of the trees) appears on at most 2 relevant edges. And, if it appears on 2 edges then those 2 edges are the 2 representatives of some 2-edge of B. Since we constructed P in such a way that at most one representative of any edge in B is included in P, each surplus color appears at most once in P. Thus P is a rainbow path. □

Lemma 5.24. *For any pair of vertices $v_1, v_2 \in V(G)$, there is a rainbow path between v_1 and v_2 in G.*

Proof. Consider any pair of vertices $v_1, v_2 \in V(G)$. If both $v_1, v_2 \in V(G) \setminus L_S$, then we are done by Lemma 5.23. Hence assume without loss of generality that $v_1 \in L_S$. Let e_1 be the edge incident on v_1 that is colored g_1, and let a be the other endpoint of e_1. If $v_2 \in L_S$, let e_2 be the edge incident on v_2 that is colored g_2, and let b be the other endpoint of e_2. If $v_2 \notin L_S$, let $b = v_2$. We know that there is a rainbow path P from a to b that uses only colors in $[f+t]$ due to Lemma 5.23. We define a path P' as follows. If $v_2 \in L_S$, then $P' := v_1 a P b v_2$; otherwise, i.e., if $v_2 \notin L_S$, then $P' := v_1 a P v_2$. Since edge $v_1 a$ is colored with $g_1 = f+t+1$, edge bv_2 is colored with $g_2 = f+t+2$, and path P uses only colors in $[f+t]$, we have that the path P' is indeed a rainbow path between v_1 and v_2. □

5.2.3 Take 3: $\mathsf{rc} \leq f+2$

In Take 2, we gave 2 surplus colors to each tree. Here, we give only 1 surplus color to each tree and thereby reduce the number of colors used. But the analysis has to be much more tighter to make the proof work with 1 surplus color per tree. This makes the proof much more technical and lengthy.

Similar to Take 2, we now fix a maximum induced forest $\mathcal{F}$ of G. But here, we fix a maximum induced forest with an additional property. Let $\mathcal{F}$ be the maximum induced forest that has the smallest number of connected components (trees) out of all the maximum induced forests of G. Now that $\mathcal{F}$ is fixed, we define $F, S, H, V_T, E_1, E_2, f_T$, tree vertices, non-tree vertices, 1-edges, 2-edges, representatives, foots, and skeleton the same way as in Take 2. But the selection of skeleton B is done in a more involved way here. Given a skeleton B with root r, we define **level** of each node v (denoted by $\ell_B(v)$) as its distance (in terms of number of vertices) to r in B. Note that the level of root r is 1 as per this definition. For a skeleton B, we define its **configuration vector** as the following vector:

$$\langle\ |E_2(B)|, \Sigma_{v:\ell_B(v)=1}\mathsf{deg}_B(v), \Sigma_{v:\ell_B(v)=2}\mathsf{deg}_B(v), \cdots, \Sigma_{v:\ell_B(v)=|V|}\mathsf{deg}_B(v)\ \rangle$$

where $\mathsf{deg}_B(v)$ is the total degree (in-degree + out-degree) of v in B. We now fix a skeleton B such that it has the lexicographically highest configuration vector out of all possible skeletons. We also define L_S and B_1 the same way as in Take 2, i.e., L_S is the set of vertices in S that are leaves of B, and $B_1 = B[V(B) \setminus L_S]$. Let $\tilde{B_1}$ be the underlying undirected tree of B_1. We note that since the first element of the configuration vector is the number of 2-edges, Lemma 5.17, Corollaries 5.18 and 5.19, and Lemma 5.21 from Take 2 holds for B here as well.

We define a mapping h from G to H as follows. For a vertex v in $V(G)$, if $v \in S$ define $h(v) := v$, otherwise (i.e., if $v \in F$) define $h(v) := x_T$, where $T \in \mathcal{T}$ is the tree containing v. For a non-tree edge uv in G, we define $h(e)$ to be the edge $h(u)h(v)$. For a vertex subset U of $V(G)$, we define $h(U)$ to be $\bigcup_{a \in U} h(a)$. For an edge subset E' of $E(G)$, we define $h(E')$ to be $\{h(e) : e \in E' \text{ and } e \text{ is a non-tree edge}\}$. For a subgraph G' of G, we define $h(G')$ as the subgraph of H with vertex set $h(V(G'))$ and edge set $h(E(G'))$. We also define a mapping g from H to G as follows. For a vertex v in $V(H)$, we define $g(v)$ to be $\{v\}$ if $v \in S$, and to be $V(f_T(v))$ otherwise (i.e., if $v \in V_T$). For an edge e in H, we define $g(e)$ to be the set of representatives of e. For a vertex or edge set

A of H, we define $g(A)$ to be $\bigcup_{a \in A} g(a)$. For a subgraph H' of H, we define $g(H')$ as the subgraph of G with vertex set $g(V(H'))$ and edge set $g(E(H')) \cup \bigcup_{x_T \in V(H') \cap V_T} E(T)$.

Let the palette of colors be $\{1, 2, \cdots, f+2\}$. We call colors $f+1$ and $f+2$ as the **global surplus colors**, denoted by g_1 and g_2. We reserve g_1 and g_2 to color the edges incident on L_S. Now, we will give a coloring of some edges of G using colors $\{1, 2, \cdots, f\}$ such that there is a rainbow path between every pair of vertices in $V(G) \setminus L_S$. We give our coloring procedure as a list of **coloring rules**.

For $a, b \in V(G) \setminus L_S$, let Q_{ab} denote the unique path in B_1 between $h(a)$ and $h(b)$. For each pair of vertices $a, b \in V(G) \setminus L_S$, we will maintain a subgraph P_{ab} of G. Each P_{ab} is initialized to $\emptyset$. After the application of each coloring rule, we will apply a path rule for each pair $a, b \in V(G) \setminus L_S$, which may add some new colored edges to P_{ab}. We say that an edge in B_1 is colored, if its representatives in G are colored (we will make sure that for a 2-edge, either both representatives are colored or both are uncolored at any point of time). Whenever an edge in B_1 gets colored by a coloring rule and if it is in Q_{ab}, we make sure that we add exactly one of its representatives to P_{ab} in the proceeding path rule (there is one exception in Coloring rule 11 that will be dealt through the addition of some *shortcut edges*; this will be explained later in Lemma 5.46). Whenever a tree T has 2 edges of P_{ab} incident on any 2 vertices u and v in T, we add the path T_{uv} to P_{ab}. And, whenever a tree T with a (or b) in T has an edge of P_{ab} incident on a vertex u of T, we add the path T_{ua} (or T_{ub}) to P_{ab}. And if both a and b are in the same tree T, then we add the path T_{ab} to P_{ab}. Thus, when all the coloring rules and path rules have been applied, we will have that for all $a, b \in V(G) \setminus L_S$, P_{ab} is a path between a and b. We will prove that P_{ab} is in fact a rainbow path, in Lemma 5.46. Once we have this, the rainbow connectivity is then extended to the whole of G, using global surplus colors, as in Take 2.

We will maintain the following invariant so that P_{ab} is a rainbow path at the end of the coloring procedure.

Invariant 1. *For each pair $a, b \in V(G) \setminus L_S$, no two edges in P_{ab} have the same color.*

Coloring rule 1. *Color all the edges in $\mathcal{F}$ with distinct colors $1, 2, \cdots, f-t$.*

Now, we proceed on to give the coloring rules and the path rules.

Path rule 1. *For each $a, b \in V(G) \setminus L_S$: If a and b are in the same tree T for some $T \in \mathcal{T}$, then add the path T_{ab} to P_{ab}.*

For each tree $T \in \mathcal{T}$, we designate a color in $[f-t+1, f]$ as its **surplus color**, denoted by $s(T)$; more specifically, the surplus color of i^{th} tree in $\mathcal{T}$ is defined as the color $f-t+i$.

Coloring rule 2. *For each 1-edge $\overrightarrow{uv}$ in B: if u is a tree vertex, then color $(uv)_1$ with $s(f_T(u))$; otherwise, i.e., if u is not a tree vertex, by Lemma 5.21, there is at least one child of u that is a tree vertex; pick one such tree vertex x_T and color $(uv)_1 = uv$ with color $s(T)$.*

Path rule 2. *Do the following for each $a, b \in V(G) \setminus L_S$. For each 1-edge e in Q_{ab}, add $(e)_1$ to P_{ab}. Next, we add edges inside some trees as follows.*
If for some tree T, $a \in V(T)$ and there is a 1-edge ux_T in Q_{ab}: let w be the foot of edge

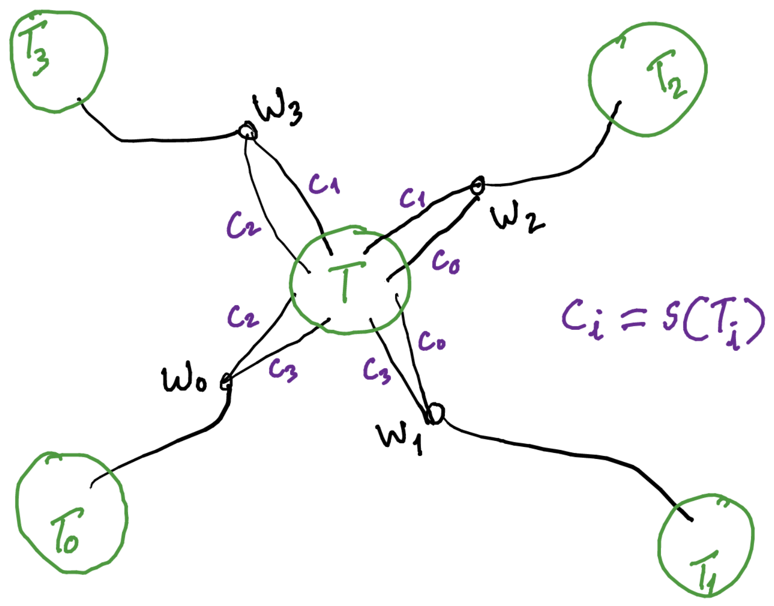


Figure 5.2: Illustration of Coloring rule 3 applied on a tree vertex x_T with 2-edge degree 4

ux_T in T; add the path T_{wa} to P_{ab}.
If for some tree T, $b \in V(T)$ and there is a 1-edge ux_T in Q_{ab}: let w be the foot of edge ux_T in T, add the path T_{wb} to P_{ab}.
If for some tree T, there are 2 1-edges ux_T and vx_T in Q_{ab}: let w be the foot of edge ux_T in T and z be the foot of edge vx_T in T; add the path T_{wz} to P_{ab}.

Note that so far, each color is used at most for one edge in G and hence Invariant 1 is satisfied. We will also maintain the following invariant throughout the coloring procedure.

Invariant 2. *For any 2-edge in B, either both representatives of it are colored or both are uncolored.*

Since no representatives of 2-edges have been colored till now, the invariant holds as of now. We will not explicitly prove this invariant after each rule, as it will be easy to observe from the coloring rules. A vertex in B_1 is said to be *completed* if all the incident edges on it in B_1 are colored; and is said to be *incomplete* otherwise. For a colored edge $e \in E(G)$, we define $c(e)$ to be the color of e. For a subgraph G' of G, we define $c(G')$ to be the set of colors used in $E(G')$. We call the number of 2-edges of B_1 incident on a vertex as the 2-*edge degree* of it in B_1. For vertices u, v, the connected component of $B_1 \setminus u$ (which is a subtree of B_1) containing v is denoted by $\mathtt{ST}(u, v)$. For u and v in B_1, the closest (breaking ties arbitrarily) tree vertex to v in $\mathtt{ST}(u, v)$ in $\tilde{B_1}$ is denoted by $\mathtt{CT}(u, v)$. Note that at least one tree vertex exist in $\mathtt{ST}(u, v)$ because all leaves of B_1 are tree vertices by Corollary 5.19. Also note that if v is a tree vertex, then $\mathtt{CT}(u, v) = v$. We also define $\mathtt{STG}(u, v) := g(\mathtt{ST}(u, v))$.

Coloring rule 3. *For each tree vertex x_T with 2-edge degree at least 4 (see Figure 5.2 for an illustration): Let $w_0, w_1, w_2, \cdots, w_{q-1}$ be the other end-points of the 2-edges incident on x_T. For $i \in [0, q-1]$, let $x_{T_i} := \texttt{CT}(w_T, w_i)$, and let $c_i := s(T_i)$. For each $i \in [0, q-1]$, color $(x_T w_i)_1$ with $c_{((i+2) \mod q)}$ and $(x_T w_i)_2$ with $c_{((i+3) \mod q)}$.*

The following lemma follows from the way in which we have colored the edges incident on x_T in Coloring rule 3.

Lemma 5.25. *For each tree vertex x_T on which Coloring rule 3 has been applied as above, for distinct $i, j \in [0, q-1]$, there is a rainbow path from w_i to w_j in G using only the colors from $(\{c_0, c_1, \cdots, c_{q-1}\} \setminus \{c_i, c_j\}) \cup c(T)$. Also, for any $i \in [q-1]$ and some $u \in V(T)$, there is a rainbow path in G from u to w_i that uses only the colors from $(\{c_0, c_1, \cdots, c_{q-1}\} \setminus \{c_i\}) \cup c(T)$.*

Proof. Let u_i and v_i be the endpoints in T of $(x_T w_i)_1$ and $(x_T w_i)_2$ respectively for each $i \in \{0, 1, \ldots, q-1\}$. First, we prove that there is a rainbow path from w_i to w_j with the required colors as claimed by the lemma. Suppose for the sake of contradiction that there was no such path. Then the path $P := w_i u_i T_{u_i u_j} u_j w_j$ is either not a rainbow path or uses a color that is not in $(\{c_0, c_1, \cdots, c_{q-1}\} \setminus \{c_i, c_j\}) \cup c(T)$. We know that the path $T_{u_i u_j}$ uses only colors from $c(T)$ and is rainbow colored, the edge $w_i u_i$ is colored $c_{(i+2) \mod q}$, and $u_j w_j$ is colored $c_{(j+2) \mod q}$. Also $c_{(i+2) \mod q} \neq c_{(j+2) \mod q}$ as $(i+2) \mod q \neq (j+2) \mod q$ for distinct $i, j \leq q$ and $q \geq 4$. Furthermore, $c_{(i+2) \mod q} \neq c_i$ and $c_{(j+2) \mod q} \neq c_j$ as $(i+2) \mod q \neq i$ for $i \leq q$ and $q \geq 4$. Thus the only possibility is that $c_{(i+2) \mod q} = c_j$ or $c_{(j+2) \mod q} = c_i$. Without loss of generality assume that $c_{(i+2) \mod q} = c_j$. That means $(i+2) \mod q = j$. We also know that the path $P' := w_i v_i T_{v_i v_j} v_j w_j$ is either not a rainbow path or uses a color that is not in $(\{c_0, c_1, \cdots, c_{q-1}\} \setminus \{c_i, c_j\}) \cup c(T)$. We know that the path $T_{v_i v_j}$ uses only colors from $c(T)$ and is rainbow colored, the edge $w_i v_i$ is colored $c_{(i+3) \mod q}$, and $v_j w_j$ is colored $c_{(j+3) \mod q}$. Also $c_{(i+3) \mod q} \neq c_{(j+3) \mod q}$, $c_{(i+3) \mod q} \neq c_i$, and $c_{(j+3) \mod q} \neq c_j$. Furthermore, $c_{j+3} \mod q = c_{(((i+2) \mod q)+3) \mod q = c_{(i+1)} \mod q \neq c_{i+3} \mod q}$. Thus P' is a rainbow path and uses only the colors in $(\{c_0, c_1, \cdots, c_{q-1}\} \setminus \{c_i, c_j\}) \cup c(T)$. Hence, we have a contradiction.

Next, we prove the second part of the lemma, i.e., we prove that there is a rainbow path from u to w_i with the required colors as claimed by the lemma. Suppose for the sake of contradiction that there was no such path. Then the path $P := w_i u_i T_{u_i u}$ is either not a rainbow path or uses a color that is not in $(\{c_0, c_1, \cdots, c_{q-1}\} \setminus \{c_i\}) \cup c(T)$. We know that the path $T_{u_i u}$ uses only colors from $c(T)$ and is rainbow colored, and that the edge $w_i u_i$ is colored $c_{(i+2) \mod q}$. Also $c_{(i+2) \mod q} \neq c_i$. Thus P is a rainbow path and uses only the colors in $(\{c_0, c_1, \cdots, c_{q-1}\} \setminus \{c_i\}) \cup c(T)$. Hence, we have a contradiction. □

Path rule 3. *For each x_T on which Coloring rule 3 has been applied as above and for each $a, b \in V(G) \setminus L_S$ such that Q_{ab} contains x_T (we say that the rule is being applied on the pair (x_T, P_{ab})):*
Case A: x_T has 2 2-edges incident in Q_{ab}.
Let w_i and w_j be the neighbors of x_T in Q_{ab}. Add to P_{ab}, the rainbow path from w_i to w_j as given by Lemma 5.25.
Case B: x_T has 1 2-edge and 1 1-edge incident in Q_{ab}.
Let $x_T w_i$ be the 2-edge. Let u be the endpoint of the representative of the 1-edge in T.

There is a rainbow path from w_i to u as given by Lemma 5.25. Add this path to Q_{ab}.
Case C: x_T is an endpoint of Q_{ab} and has 1 2-edge incident in Q_{ab}.
Let w_i be the neighbor of x_T in Q_{ab}. We know one of a or b is in T. From this vertex (a or b whichever is in T) to w_i, there is a rainbow path as given by Lemma 5.25. Add this path to P_{ab}.

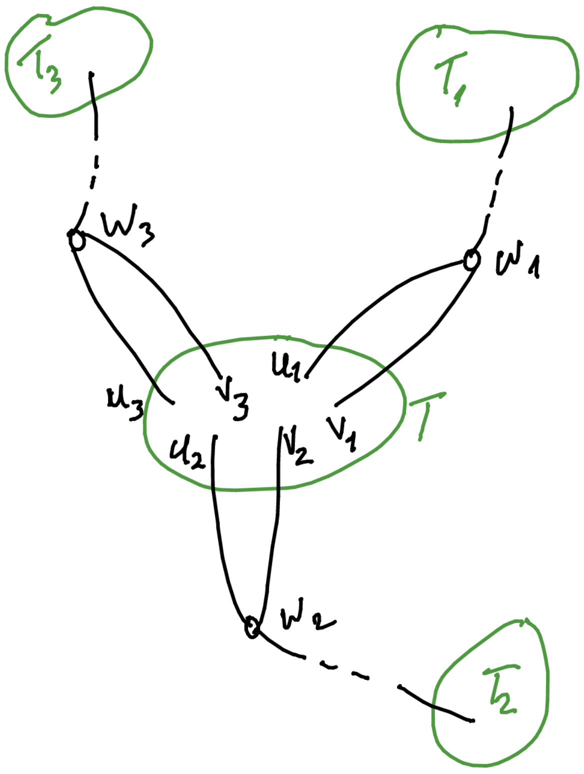
The following lemma follows from Lemma 5.25 and Path rule 3.

Lemma 5.26. *Suppose for some $a, b \in V(G) \setminus L_S$ and for some tree $T' \in \mathcal{T}$, P_{ab} contains an edge e that was colored with $s(T')$ during the application of Coloring rule 3 on some tree vertex x_T. Then, $T' \neq T$ and Q_{ab} does not intersect $\texttt{ST}(x_T, x_{T'})$.*

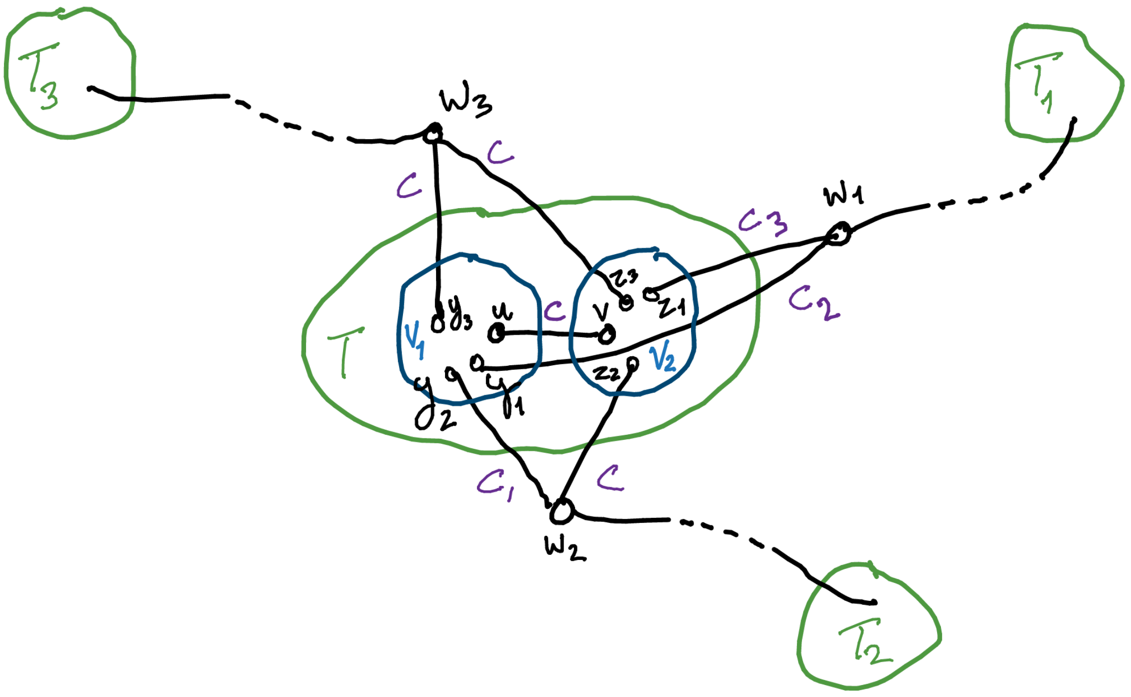
Proof. Since $s(T')$ was used during the application of Coloring rule 3 on x_T, the vertex $x_{T'}$ should have been taken as x_{T_i} (in Coloring rule 3) for some i and $s(T')$ was taken as c_i (in Coloring rule 3). Since $T_i \neq T$, it is clear that $T' \neq T$. Suppose Q_{ab} intersects $\texttt{ST}(x_T, x_{T'})$ for the sake of contradiction. Then the color c_i was not used in Path rule 3 according to Lemma 5.25. That means e was not colored with c_i, which is a contradiction. $\square$

Lemma 5.27. *Invariant 1 is not violated during Path rule 3.*

Proof. Suppose Invariant 1 is violated during the application of Path rule 3 on the pair (x_T, P_{ab}) as above. Then there exist edges e and e' in P_{ab} having the same color. We can assume without loss of generality that e was colored during the application of Coloring rule 3 on x_T. Then either $e \in E(T)$ or e is a representative of $w_i x_T$ for some $i \in [0, q-1]$. Since each color in $c(T)$ have been used only in one edge in G, we have that $h(e) = w_i x_T$ for some $i \in [0, q-1]$ and hence $c(e) = s(T_j)$ for some $j \in [0, q-1] \setminus i$. Also Q_{ab} does not intersect $\texttt{ST}(x_T, x_{T_j})$ by Lemma 5.26. Since application of Path rule 3 on (x_T, P_{ab}) added a rainbow path to P_{ab}, the edge e' was not added during this application. Since each color in $c(F)$ is used only for one edge in G so far, we also know that e' was not added during Path rule 1. Hence the following cases are exhaustive and in each case we prove a contradiction.
Case I: e' was added during the application of Path rule 3 on $(x_{T'}, P_{ab})$ for some tree $T' \neq T$.
Since P_{ab} contains e', we have that Q_{ab} contains $h(e')$. Since e' was added during the application of Path rule 3 on $(x_{T'}, P_{ab})$, either $e' \in E(T')$ or $h(e')$ is incident on $x_{T'}$, and hence $x_{T'}$ is in Q_{ab}. Since Q_{ab} does not intersect $\texttt{ST}(x_T, x_{T_j})$, we have that $x_{T'}$ is not in $\texttt{ST}(x_T, x_{T_j})$. Then $\mathsf{dist}_{B_1}(x_{T'}, x_T) < \mathsf{dist}_{B_1}(x_{T'}, x_{T_j})$. But then during the application of Coloring rule 3 on $x_{T'}$, the color $s(T_j)$ would never be used as $x_{T_j} \neq \texttt{CT}(x_{T'}, v)$ for any vertex v. Thus, the color of e' is not $s(T_j)$. But we know that $c(e') = c(e) = s(T_j)$, a contradiction.
Case II: e' was added during the application of Path rule 2 on P_{ab}.
This means e' is the representative of a 1-edge and was colored during Coloring rule 2. Since e' is colored with $s(T_j)$, we have that $h(e')$ should either be the outgoing edge of x_{T_j} or the outgoing edge of the parent of x_{T_j}, from Coloring rule 2. This implies that $h(e')$ is in $\texttt{ST}(x_T, x_{T_j})$, as the parent of x_{T_j} is a non-tree edge. But then Q_{ab} does not contain $h(e')$ as Q_{ab} does not intersect $\texttt{ST}(x_T, x_{T_j})$. Thus P_{ab} does not contain e', which is a contradiction. $\square$

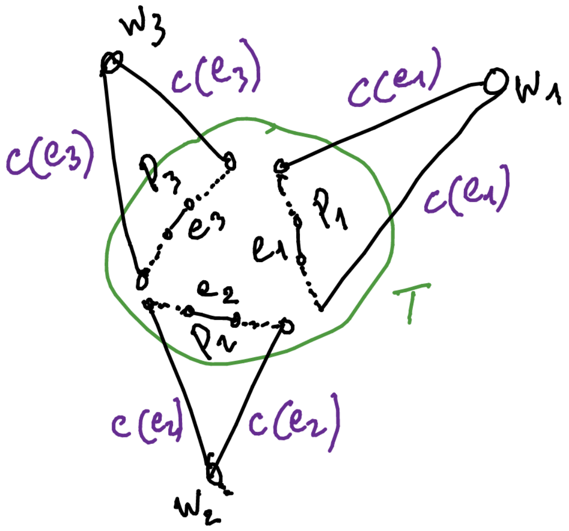


(a) A scenario in which Coloring rule 4 is applicable on x_T

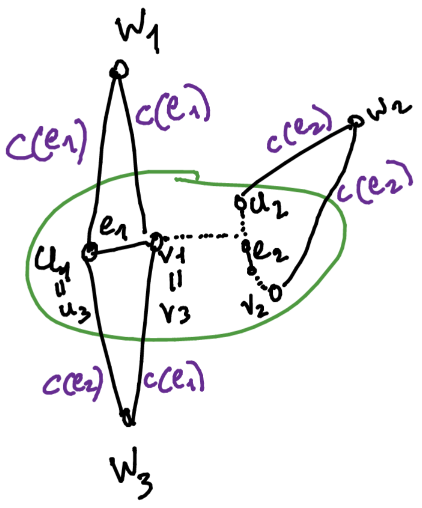


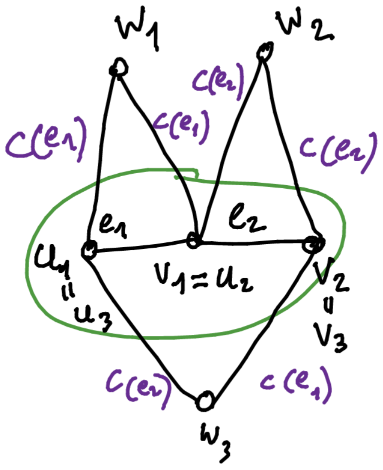
(b) Coloring rule 4: Case 1

Figure 5.3: Illustration of Coloring rule 4 (continued on next page)



(c) Coloring rule 4: Case 2





(d) Coloring rule 4: Case 3: scenario 1 **(e)** Coloring rule 4: Case 3: scenario 2

Coloring rule 4. *For each tree vertex x_T with 2-edge degree exactly 3 (see Figure 5.3(a)): let w_1, w_2, and w_3 be the other end-points of the 3 2-edges incident on x_T; for $i \in \{1,2,3\}$, let $x_{T_i} = \mathtt{CT}(x_T, w_i)$, let u_i and v_i be the foots of $x_T w_i$ in T, let $P_i := T_{u_i v_i}$, and let $c_i := s(T_i)$.*
<u>*Case 1*</u>*. (See Figure 5.3(b)) If there exist edge uv in T such that the cut (V_1, V_2) induced by uv in T is such that for all $i \in \{1,2,3\}$, $|V_1 \cap \{u_i, v_i\}| = 1$ and $|V_2 \cap \{u_i, v_i\}| = 1$: Let y_i and z_i be the foots of $x_T w_i$ in V_1 and V_2 respectively for each $i \in \{1,2,3\}$. Let c be the color of uv. Color $y_1 w_1$ with c_3, $z_1 w_1$ with c_2, $y_2 w_2$ with c_1, $z_2 w_2$ with c, $y_3 w_3$ with c, and $z_3 w_3$ with c.*
<u>*Case 2*</u>*. (See Figure 5.3(c)) If there exist distinct edges e_1, e_2, e_3 such that $e_i \in E(P_i)$ for each $i \in \{1,2,3\}$: Color both the representatives of $x_T w_i$ with the color of e_i for each $i \in \{1,2,3\}$.*
<u>*Case 3*</u>*. If Case 1 and 2 does not apply: then there exist $i, j \in \{1,2,3\}$ such that $E(P_i) \cap E(P_j) = \emptyset$, because otherwise $E(P_1) \cap E(P_2) \cap E(P_3) \neq \emptyset$ using the Helly property of trees* [1] *and then any edge in this intersection qualifies as uv of Case 1. So, without loss of generality assume that $E(P_1) \cap E(P_2) = \emptyset$. Also, note that $E(P_3) \subseteq E(P_1) \cup E(P_2)$ because otherwise Case 2 applies. So, without loss of generality assume that $E(P_3) \cap E(P_1) \neq \emptyset$. But then $P_3 \cap P_1 = P_1$ and P_1 consists of a single edge so that Case 2 does not apply. Let this edge be e_1. Note that $e_1 = u_1 v_1$. Furthermore, at least one of the end-vertices of P_1 and P_3 coincide so that Case 2 does not apply. Thus, assume without loss of generality that $u_1 = u_3$. Let e_2 be any edge in P_2. Without loss of generality assume that v_1 is the closer among u_1, v_1 to path P_2 in T. The two possible scenarios in this case are shown in Figures 5.3(d) and 5.3(e). Color $w_1 u_1$ and $w_1 v_1$ with $c(e_1)$, $w_2 u_2$ and $w_2 v_2$ with $c(e_2)$, $w_3 u_3$ with $c(e_2)$ and $w_3 v_3$ with $c(e_1)$.*

The following lemma follows from the way in which we have colored the edges incident on x_T in Coloring rule 4.

Lemma 5.28. *For each tree vertex x_T on which Rule 4 has been applied as above, for distinct $i, j \in \{1,2,3\}$, there is a rainbow path from w_i to w_j in G that uses only the colors from $(\{c_1, c_2, c_3\} \setminus \{c_i, c_j\}) \cup c(T)$. Also, for any $i \in \{1,2,3\}$, and any $z \in V(T)$, there is a rainbow path from u to w_i, that uses only the colors from $(\{c_1, c_2, c_3\} \setminus \{c_i\}) \cup c(T)$.*

Proof. We demonstrate the required paths in each of the 3 cases of Coloring Rule 4.

<u>Case 1</u>: Between w_1 and w_2, there is the path $w_1 z_1 T_{z_1 z_2} z_2 w_2$ that uses only the colors in $c(T) \cup \{c_3, c\}$. Between w_1 and w_3, there is the rainbow path $w_1 y_1 T_{y_1 y_3} y_3 w_3$ that uses only the colors in $c(T) \cup \{c_2, c\}$. Between w_2 and w_3, there is the rainbow path $w_2 y_2 T_{y_2 y_3} y_3 w_3$ that uses only the colors in $c(T) \cup \{c_1, c\}$.

Now consider any vertex $y \in V_1$. Between y and w_1, there is the rainbow path $T_{y y_1} y_1 w_1$ that uses only the colors in $c(T) \cup \{c_2\}$. Between y and w_2, there is the rainbow path $T_{y y_2} y_2 w_2$ that uses only the colors in $c(T) \cup \{c_1\}$. Between y and w_3, there is the rainbow path $T_{y y_3} y_3 w_3$ that uses only the colors in $c(T) \cup \{c\}$.

Now consider any $z \in V_2$. Between z and w_1, there is the rainbow path $T_{z z_1} z_1 w_1$ that uses only the colors in $c(T) \cup \{c_3\}$. Between z and w_2, there is the rainbow path

[1]We use the following Helly property of trees: If $T_1, T_2, \ldots, T_k$ are subtrees of a tree T that pairwise intersect each other on at least one edge, then there is an edge of T that is common to all of $T_1, T_2, \ldots, T_k$.

$T_{zz_2}z_2w_2$ that uses only the colors in $c(T) \cup \{c\}$. Between z and w_3, there is the rainbow path $T_{zz_3}z_3w_3$ that uses only the colors in $c(T) \cup \{c\}$.

Case 2: First we show the path between w_1 and w_2. By Observation 5.22, either $T_{u_1u_2}$ or $T_{u_1v_2}$ does not contain the edge e_2. If $T_{u_1u_2}$ does not contain e_2, then the path $w_1u_1T_{u_1u_2}u_2w_2$ is a rainbow path and uses only the colors in $c(T)$; otherwise (i.e., if $T_{u_1v_2}$ does not contain e_2) then the path $w_1u_1T_{u_1v_2}v_2w_2$ is a rainbow path and uses only the colors in $c(T)$. The required paths between w_2 and w_3, and between w_1 and w_2 can be shown in a similar way.

Now for any vertex z in T, we show the required path between z and w_1. By Observation 5.22, either T_{uu_1} or T_{uv_1} does not contain the edge e_1. If T_{uu_1} does not contain e_1, then the path $T_{uu_1}u_1w_1$ is a rainbow path and uses only the colors in $c(T)$; otherwise (i.e., if T_{uv_1} does not contain e_1) then the path $T_{uv_1}v_1w_1$ is a rainbow path and uses only the colors in $c(T)$. The required paths between w_2 and z, and between w_3 and z can be shown in a similar way.

Case 3: First we show the required path between w_1 and w_2. By Observation 5.22, either $T_{v_1u_2}$ or $T_{v_1v_2}$ does not contain the edge e_2. Observe that both $T_{v_1u_2}$ and $T_{v_1v_2}$ does not contain the edge e_1 as v_1 is closer than u_1 to P_2 , as mentioned in the Coloring Rule. Hence, if $T_{v_1u_2}$ does not contain e_2, then the path $w_1v_1T_{v_1u_2}u_2w_2$ is a rainbow path that uses only the colors in $\{c(e_2), c(e_1)\} \cup c(T)$; and otherwise (i.e., if $T_{v_1v_2}$ does not contain e_2), the path $w_1v_1T_{v_1v_2}v_2w_2$ is a rainbow path that uses only the colors in $\{c(e_2), c(e_1)\} \cup c(T)$.

Since $u_1 = u_3$, there is the path $w_1u_1w_3$ between w_1 and w_3 that uses only the colors in $\{c(e_1), c(e_2)\}$. Next, we show the required path between w_3 and w_2. By Observation 5.22, either $T_{v_3u_2}$ or $T_{v_3v_2}$ does not contain the edge e_2. Let v' be the vertex in $\{u_2, v_2\}$ such that $T_{v_3v'}$ does not contain edge e_2. We show that the path $w_3v_3T_{v_3v'}v'w_2$ is the required path between w_3 and w_2. We know w_3v_3 is colored $c(e_1)$ and w_2v' is colord $c(e_2)$. So, it is sufficient to show that $c(e_1)$ and $c(e_2)$ does not appear in $T_{v_3v'}$. For this, it is sufficient to prove that e_1 and e_2 is not in $T_{v_3v'}$. Since we picked v' such that $T_{v_3v'}$ does not contain e_2, it only remains to prove that e_1 is not in $T_{v_3v'}$. Suppose for the sake of contradiction that e_1 is in $T_{v_3v'}$. That means both v_1 and u_1 are in $T_{v_3v'}$. We know that v_1 is closer than u_1 to P_2, as mentioned in the Coloring Rule. Hence, v_1 is closer than u_1 to v' in T. This also implies that v_1 is closer than u_1 to v' in $T_{v_3v'}$. Then u_1 is closer than v_1 to v_3 in $T_{v_3v'}$ and hence also in T. But then P_3 contains edges that are not in both P_1 and P_2, a contradiction.

Now consider any vertex $z \in V(T)$. For each $i \in \{1, 3\}$, let v'_i be the closer vertex among u_i, v_i to z. and let P_i be the path $T_{zv'_i}v'_iw_i$. We show that P_i is the required path from z to w_i for $i \in \{1, 3\}$. The path from z to v'_1 does not contain e_1. Also, the edge $w_1v'_1$ is colored with $c(e_1)$. Hence P_1 is a rainbow path from z to w_1 that uses only colors in $c(T)$. Now consider path P_3. First consider the case when $v'_3 = u_3 = u_1$. Then e_2 is not in $T_{zv'_3}$, because otherwise either P_3 contains edges that are not in $P_1 \cup P_2$ or u_1 is closer than v_1 to P_2. Since u_3w_3 is colored $c(e_2)$, the path P_3 satisfies the requirements. Now consider the case when $v'_3 = v_1$. Then e_1 is clearly not in $T_{zv'_3}$. Since v_3w_3 is colored $c(e_1)$, the path P_3 satisfies the requirements.

Now we show the required path from z to w_3. By Observation 5.22, either T_{zv_2} or T_{zu_2} does not contain the edge e_2. Let v'_2 be the vertex in $\{u_2, v_2\}$ such that $T_{zv'_2}$ does not contain edge e_2. Then the path $T_{zv'_2}v'_2w_2$ is the required path between z and w_2, as

$v_2'w_2$ is colored $c(e_2)$. □

Path rule 4. *For each x_T on which Coloring rule 4 has been applied as above and for each P_{ab} such that Q_{ab} contains x_T (we say that the rule is being applied on the pair (x_T, P_{ab})):*
Case A. If x_T has 2 2-edges incident in Q_{ab}:
Let w_i and w_j be the neighbors of x_T in Q_{ab}. Add to P_{ab}, the rainbow path from w_i to w_j as given by Lemma 5.28.
Case B. If x_T has exactly one 2-edge incident in Q_{ab}:
Let x_Tw_i be the 2-edge and let z be the endpoint in T of the 1-edge. Add to P_{ab}, the rainbow path from w_i to z as given by Lemma 5.28.
Case C. If x_T is an endpoint of Q_{ab} and has 1 2-edge incident in Q_{ab}:
Let w_i be the neighbor of x_T in Q_{ab}. We know one of a or b is in T. From this vertex (a or b whichever is in T) to w_i, there is a rainbow path as given by Lemma 5.25. Add this path to P_{ab}.

The following lemma follows from Lemma 5.28 and Path rule 4. The proof is similar to that of Lemma 5.26 and is omitted.

Lemma 5.29. *Suppose for some $a, b \in V(G) \setminus L_S$ and for some tree $T' \in \mathcal{T}$, P_{ab} contains an edge e that was colored with $s(T')$ during the application of Coloring rule 4 on some tree vertex x_T. Then, $T' \neq T$ and Q_{ab} does not intersect $\texttt{ST}(x_T, x_{T'})$.*

Lemma 5.30. *Invariant 1 is not violated during Path rule 4.*

Proof. Suppose for the sake of contradiction that Invariant 1 is violated during the application of Path rule 4 on the pair (x_T, P_{ab}) as above. Then there exist edges e and e' in P_{ab} having the same color. We can assume without loss of generality that e was colored during the application of Coloring rule 4 on x_T. This means $e \in E' := E(T) \cup g(\{w_1x_T, w_2x_T, w_3x_T\})$. Since, application of Path rule 4 on (x_T, P_{ab}) added a rainbow path to P_{ab}, the edge e' was not added during this application and hence $e' \notin E'$. Each color in $c(T)$ have been used only in E' so far. That means $c(e) = c(e') \notin c(T)$. Hence $e \in E' \setminus E(T) = g(\{w_1x_T, w_2x_T, w_3x_T\})$. Without loss of generality assume that $e = w_1x_T$. Now, $c(e) = s(T_j)$ where $j \in \{2, 3\}$. Without loss of generality assume that $c(e) = s(T_2)$. This also means $c(e') = s(T_2)$. That means e' was colored during Coloring rules 2, 3 or 4. Hence the following cases are exhaustive and in each case we prove a contradiction.
Case I: e' was colored during the application of Coloring rules 4 or 3 on $x_{T'}$, for some tree $T' \neq T$.
Since P_{ab} contains e', we have that Q_{ab} contains $h(e')$. Since e' was colored during the application of Coloring rules 3 or 4 on $x_{T'}$, either $e' \in E(T')$ or $h(e')$ is incident on $x_{T'}$, and hence $x_{T'}$ is in Q_{ab}. Since Q_{ab} does not intersect $\texttt{ST}(x_T, x_{T_2})$ by Lemmas 5.26 and 5.29, we have that $x_{T'}$ is not in $\texttt{ST}(x_T, x_{T_2})$. Then $\mathsf{dist}_{B_1}(x_{T'}, x_T) < \mathsf{dist}_{B_1}(x_{T'}, x_{T_2})$. But then during the application of Coloring rule 3 on $x_{T'}$, the color $s(T_2)$ would never be used as $x_{T_2} \neq \texttt{CT}(x_{T'}, v)$ for any vertex v. Thus, the color of e' is not $s(T_2)$. But we know that $c(e') = c(e) = s(T_2)$, a contradiction.
Case II: e' was colored during the application of Coloring rule 2.
This means e' is the representative of a 1-edge. Since e' is colored with $s(T_2)$, we have that

$h(e')$ should either be the outgoing edge of x_{T_2} or the outgoing edge of the parent of x_{T_2}, from Coloring rule 2. This implies that $h(e')$ is in $\texttt{ST}(x_T, x_{T_2})$, as the parent of x_{T_2} is a non-tree edge. But then Q_{ab} does not contain $h(e')$ as Q_{ab} does not intersect $\texttt{ST}(x_T, x_{T_2})$, by Lemmas 5.26 and 5.29. Thus P_{ab} does not contain e', which is a contradiction. □

Coloring rule 5. *For each non-tree vertex u with degree at least 3 in B_1 (see Figure 5.4 for examples):*
Let q be the number of children of u (note that $q \geq 2$ as degree of u is at least 3), let $u_1, u_2, \cdots u_q$ be the children of u and let x_{T_i} be $\texttt{CT}(u, u_i)$. Let $\overrightarrow{uv}$ be the outgoing edge from u in B_1. If uv is a 1-edge, due to Rule 2, we know that there exist an $i \in [q]$ such that, u_i is a tree vertex (and hence $T_i = f_T(u_i)$), and uv is colored with $s(T_i)$. Hence, if uv is a 1-edge, assume without loss of generality that u_1 is a tree vertex (and hence $x_{T_1} = u_1$) and that uv is colored with $s(T_1)$.
If u_1u is a 2-edge (then u_1 is a tree vertex and hence $T_1 = f_T(u_1)$) and is uncolored, then color $(u_1u)_1$ with $s(T_1)$ and $(u_1u)_2$ with $s(T_2)$. For each $2 \leq i \leq q$, if u_iu is a 2-edge (then u_i is a tree vertex and hence $T_i = f_T(u_i)$) and is uncolored, then color both its representatives with $s(T_i)$. Let $\overrightarrow{uv}$ be the outgoing edge from u. If uv is a 2-edge and uncolored, then color $(uv)_1$ with $s(T_1)$ and $(uv)_2$ with $s(T_2)$.

Path rule 5. *For each non-tree vertex u on which Coloring rule 5 has been applied as above and for each P_{ab} such that Q_{ab} contains u (we say that the rule is being applied on the pair (u, P_{ab})):*

There are two parts to this rule:

Part 1*: If Q_{ab} contains edge u_1u and u_1u is colored during the application of Coloring rule 5 on u: if the other neighbor (if any) of u in Q_{ab} is u_2 then add $(u_1u)_1$ (which has color $s(T_1)$) to P_{ab}; otherwise add $(u_1u)_2$ (which has color $s(T_2)$) to P_{ab}.*

For each $i \in [2, q]$, if Q_{ab} contains edge u_iu and u_iu is colored during the application of Coloring rule 5 on u, add $(u_iu)_1$ to P_{ab}.

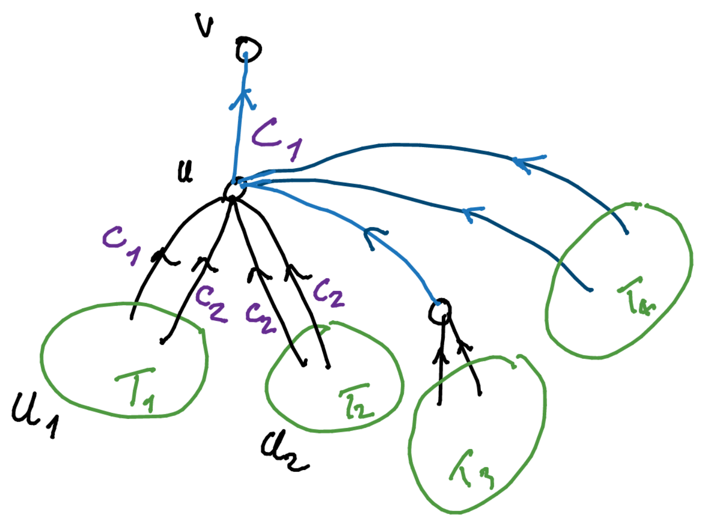
If Q_{ab} contains edge uv and uv is colored during the application of Coloring rule 5 on u: if the other neighbor (if any) of u in Q_{ab} is u_1 and u_1u is a 1-edge, then add $(uv)_2$ (which has color $s(T_2)$) to P_{ab}; otherwise add $(uv)_1$ (which has color $s(T_1)$) to P_{ab}.

Part 2*: For each tree vertex x_T such that the degree of x_T in $h(P_{ab})$ became 2 during the addition of above edges in Part 1, let x and y be the endpoints in T of the 2 edges of P_{ab} incident on T; add T_{xy} to P_{ab}.*

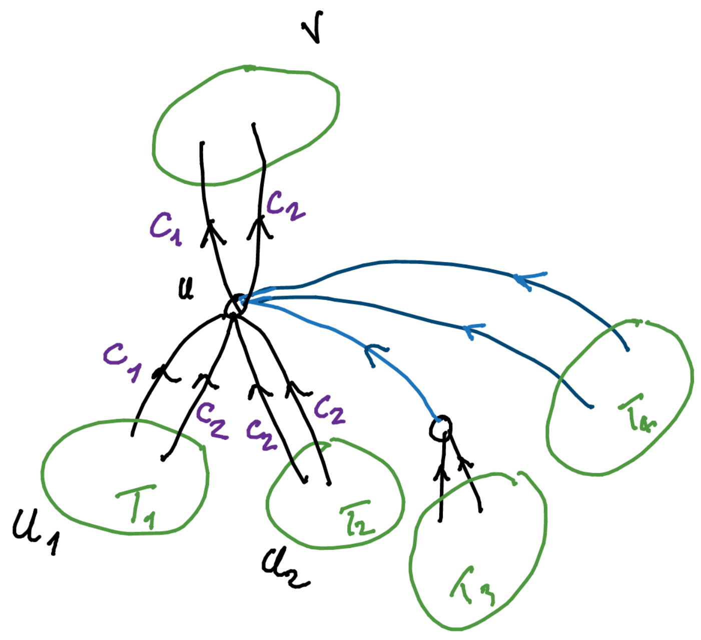
For each tree vertex $x_T \in \{h(a), h(b)\}$ such that the degree of x_T in $h(P_{ab})$ became 1 during the addition of above edges in Part 1: Let x be the endpoint in T of the edge of P_{ab} incident on T. If $x_T = h(a)$, add T_{ax} to P_{ab}; otherwise (i.e., if $x_T = h(b)$), add T_{bx} to P_{ab}.

Lemma 5.31. *Invariant 1 is not violated during Path rule 5.*

Proof. Suppose Invariant 1 is violated during the application of Path rule 5 on the pair (u, P_{ab}) as above. Then there exist edges e and e' in P_{ab} having the same color. We can assume without loss of generality that e was colored during the application of Coloring rule 5 on u. Suppose e was added during Part 2 of Path rule 5. We know the trees in which we add the path in Part 2 were *incomplete* before the application of Coloring rule 5, and hence the colors inside them were not used anywhere else so far. Thus, the

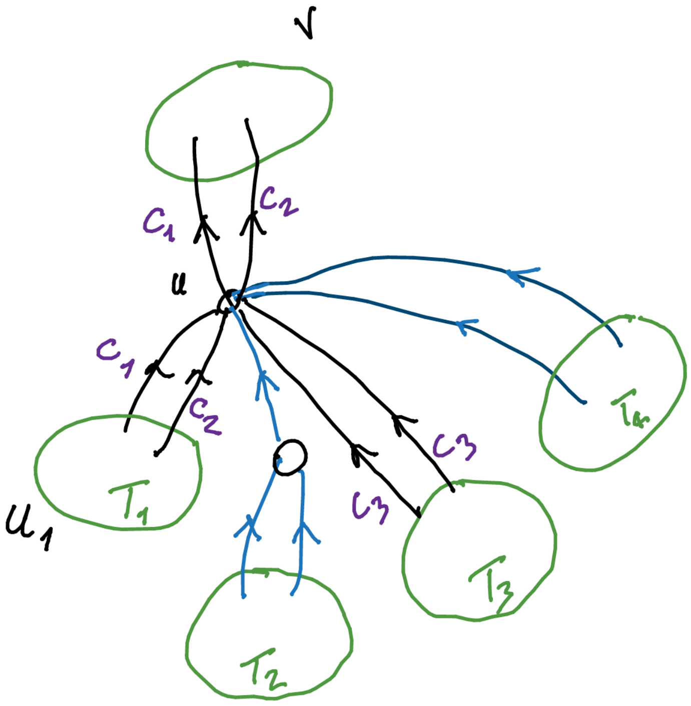


(a) Coloring rule 5: example 1



(b) Coloring rule 5: example 2

Figure 5.4: Three examples of Coloring rule 5. Here $c_i = s(T_i)$. The blue edges are the edges that were already colored before the application of the rule. The figure is continued on the next page.



(c) Coloring rule 5: example 3

color of e is unique, in particular $c(e') \neq c(e)$, a contradiction. Thus, the edge e was not added during Part 2. Then e was added during Part 1 and hence $c(e) = c(e') = s(T_i)$ for some $i \in [q]$. Then e' was colored during one of Coloring rules 5, 4, 3, or 2.
Case I. e' was colored during the Coloring rule 5
Note that during the application of Path rule 5 on (u, P_{ab}), we have added at most 2 edges to P_{ab}. And, if we have added 2 edges, they are of different color. Thus e' was not added to P_{ab} during the application of Path rule 5 on (u, P_{ab}) and hence was not colored during the application of Coloring rule 5 on u. So, e' was colored during the application of Coloring rule 5 on some non-tree vertex $u' \neq u$. Notice that for any tree $T \in \mathcal{T}$, $s(T)$ is used during application of Coloring rule 5 only when the rule is applied to an ancestor of x_T in B_1. Hence both u and u' are ancestors of x_{T_i}. Without loss of generality, assume that u' is closer than u to x_{T_i}. Then, u cannot have any tree vertices as children because otherwise $x_{T_i} \neq \texttt{CT}(u, u_i)$. Then, the only edges colored during the application of Coloring Rule 5 on u, are the representatives of uv. Thus $h(e) = uv$.
Case I.a. $e = (uv)_2$.
Then, the edge $(uv)_2$ is added during application of Path rule 5 on (u, P_{ab}). This implies that the neighbor of u in Q_{ab} is u_1, by Path rule 5. Since $u' \in Q_{ab}$, we have that $T_i = T_1$, and hence $c(e) = s(T_1)$. But we know that $e = (uv)_2$ is colored with $s(T_2) \neq s(T_1)$, by Coloring rule 5. Thus, we have a contradiction.
Case I.b. $e = (uv)_1$.
Then, the edge $(uv)_1$ is added during application of Path rule 5 on (u, P_{ab}). This implies that either the neighbor of u in Q_{ab} is u_2, or uu_1 is a 2-edge, by Path rule 5. But uu_1 cannot be a 2-edge as both u and u_1 are non-tree vertices. (Recall that we said all

children of u are non-tree vertices in the current case). Hence the neighbor of u in Q_{ab} is u_2. Since $u' \in Q_{ab}$, we have that $T_i = T_2$, and hence $c(e) = s(T_2)$. But we know that $e = (uv)_1$ is colored with $s(T_1) \neq s(T_2)$, by Coloring rule 5. Thus, we have a contradiction.
Case II. e' was colored during the Coloring rules 4 or 3.
Let T' be the tree on which e' is incident. Then e' was colored with $s(T_i)$ during the application of Coloring rules 4 or 3 on $x_{T'}$. Then Q_{ab} does not intersect $\texttt{ST}(T', T_i)$ due to Lemmas 5.29 and 5.26. Since $x_{T_i} = \texttt{CT}(u, u_i)$, ther is no other tree-vertex in the path from u to x_{T_i}. Thus, u is in $\texttt{ST}(T', T_i)$. Hence, we have that u is not in Q_{ab}. We know that e is adjacent on u as every edge colored during the application of Coloring rule 5 on u is incident on u. But then $e \notin P_{ab}$ as u is not in Q_{ab}. This is a contradiction.
Case III. e' was colored during the Coloring rule 2.
This means that e' is a 1-edge. The only possibility for 1-edge e' to have color $s(T_i)$ is either $e' = uv$ or that $T_i = T_2$ and e' is on the path between x_{T_2} and u.
Case III.a. $h(e') = uv$.
In the case when uv is a 1-edge, we selected u_1 during the Coloring rule 5 in such a way that $c(uv) = s(T_1)$. Thus $c(e) = c(e' = uv) = s(T_1)$. That means $e = (u_1u)_1$. But since uv is in Q_{ab}, we would have added $(u_1u)_2$ and not $(u_1u)_1$ to P_{ab} during Path rule 5. Thus we have a contradiction.
Case III.b. $h(e') \neq uv$.
Then $T_i = T_2$, and e' is on the path between x_{T_2} and u. Since x_{T_2} is not a child of u, the only possibility for e to have color $s(T_2)$ is if $e = (u_1u)_2$. We know Q_{ab} contains both u_1 and u_2. In that case, we would have added $(u_1u)_1$ and not $(u_1u)_2$ to P_{ab} during Path rule 5. □

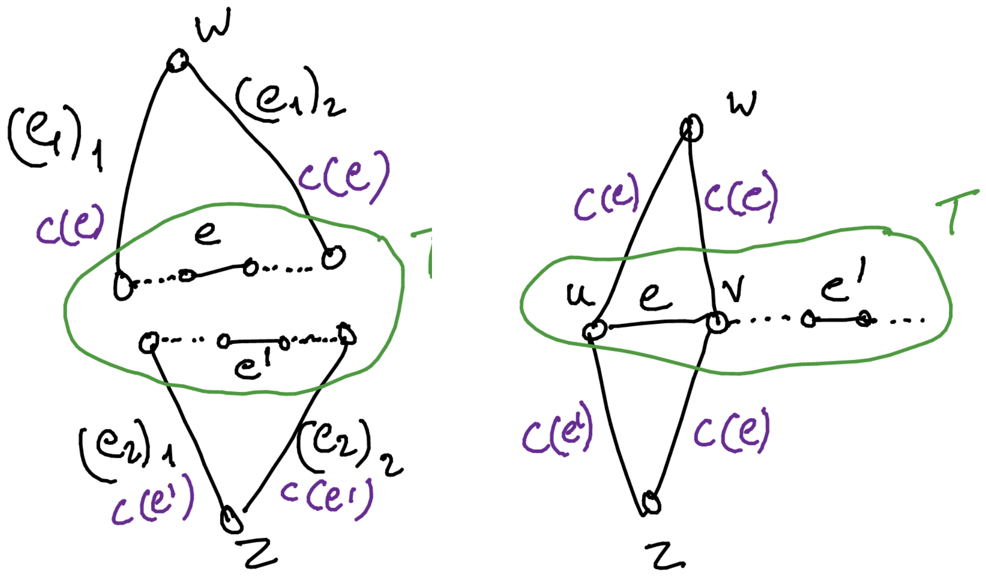
For a 2-edge e in B incident on tree vertex x_T, we define its **foot path** as the path between the foots of e in T.

Coloring rule 6. *For each incomplete tree vertex x_T having 2-edge degree exactly 1: let e be the only 2-edge incident on x_T, pick an edge e_1 in the foot path of e, color the representatives of e with the color of e_1.*

Path rule 6. *For each tree vertex x_T on which Coloring rule 6 has been applied as above and for each P_{ab} such that Q_{ab} contains $h(e)$ (we say that the path rule is being applied on the pair (x_T, P_{ab})):*
If $a \in V(T)$, let $w := a$, and if $b \in V(T)$ let $w := b$. (Note that both a and b cannot be in T as Q_{ab} contains $h(e)$). If $a, b \notin V(T)$ then there is an edge $e_2 \neq e$ of Q_{ab} incident on x_T; and since e is the only 2-edge incident on x_T, the edge e_2 is a 1-edge; let w be the endpoint of $(e_2)_1$ in T. By Observation 5.22, there is a path in T that excludes e_1, from w to one of the foots of e. Let this foot be z. Add the path in T between w and z to P_{ab}. Also add to P_{ab}, the representative of e, having z as its endpoint in T.

Lemma 5.32. *Invariant 1 is not violated during Path rule 6.*

Proof. The edges added to P_{ab} during the application of Path rule 6 to (x_T, P_{ab}) were all having colors from $c(T)$. None of the colors in $c(T)$ were used before anywhere outside T. The path from w to z does not contain e_1 and the color of the added representative of e is $c(e_1)$. Thus the edges added were all having distinct colors and these colors were not used in P_{ab} before. □



(a) Coloring rule 7 Case 1. **(b)** Coloring rule 7 Case 2.

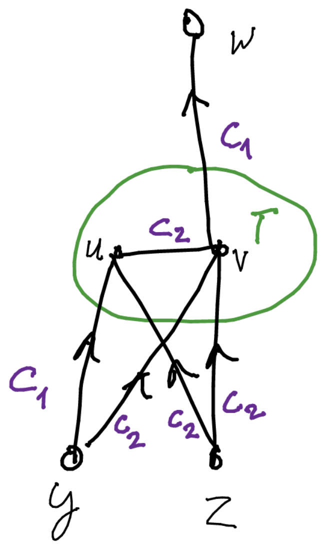
Figure 5.5: Coloring rule 7

Coloring rule 7. *For each tree vertex x_T such that the 2-edge degree of x_T is exactly 2 and T contains at least 2 edges:*
Let e_1 and e_2 be the 2-edges incident on x_T, Let w and z be the other end points of e_1 and e_2 respectively. Let P_1 and P_2 be the foot paths of e_1 and e_2 respectively.
Case 1: $|E(P_1 \cup P_2)| \geq 2$. (See Figure 5.5(a)).
Pick distinct edges e and e' from P_1 and P_2 respectively. If $(e_1)_1$ and $(e_1)_2$ are uncolored, color them with color of e and if $(e_2)_1$ and $(e_2)_2$ are uncolored, color them with color of e'.
Case 2: Case 1 does not hold.(See Figure 5.5(b)).
Clearly then P_1 and P_2 both are a single edge and they are the same edge. Let this edge be $e = uv$. Pick any other edge e' in T. Without loss of generality assume that e' is closer to v than u in T. If uw and vw are uncolored then color them with color of e. If uz and vz are uncolored, color them with colors of e' and e respectively.

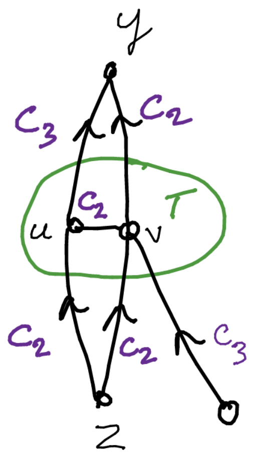
The following lemma follows straightforward from the above coloring rule.

Lemma 5.33. *Consider a tree vertex x_T on which Coloring rule 7 has been applied as above. There is a rainbow path in G from w to z using only the colors in $c(T)$. Also, from any vertex $x \in V(T)$, there is a rainbow path to both w and z using only the colors in $c(T)$.*

Path rule 7. *For each tree vertex x_T on which Coloring rule 7 has been applied as above and for each P_{ab} such that Q_{ab} contains at least one of e_1 and e_2 (we say that the path rule is being applied on the pair (x_T, P_{ab})):*
If Q_{ab} contains both e_1 and e_2 then let $y_1 := w$ and $y_2 := z$. If Q_{ab} contains only e_1 and



(a) Coloring rule 8 Case 1. Here $c_1 = s(T)$ and $c_2 = c(uv)$.



(b) Coloring rule 8 Case 2. Here $c_2 = c(uv)$ and c_3 is the color of the representative of an arbitrarily chosen 1-edge incident on x_T.

Figure 5.6: Coloring rule 8

not e_2 then let $y_1 := w$. If Q_{ab} contains only e_2 and not e_1 then let $y_1 := z$. If $a \in V(T)$, let $y_2 := a$, and if $b \in V(T)$ let $y_2 := b$. (Note that both a and b cannot be in T as Q_{ab} contains e_1 or e_2). If $a, b \notin V(T)$ and only one of e_1, e_2 is in Q_{ab} then there is an edge $e'' \notin \{e_1, e_2\}$ incident on x_T in Q_{ab}; and since e_1 and e_2 are the only 2-edges incident on x_T, the edge e'' is a 1-edge; let y_2 be the endpoint of $(e'')_1$ in T. Add to P_{ab} the path between y_1 and y_2 given by Lemma 5.33.

Lemma 5.34. *Invariant 1 is not violated during Path rule 7.*

Proof. The path added to P_{ab} during the application of Path rule 7 on x_T uses only colors from $c(T)$ and the path added is a rainbow path. The edges colored with colors from $c(T)$ could not have been added in any application of any path rule so far. Thus the invariant is not violated. □

Coloring rule 8. *For each incomplete tree vertex x_T having degree at least 3 in B_1: We can assume that Coloring rules 3, 4, 6, 7 are not applicable on x_T as otherwise x_T would have been completed. If x_T had at least 3 2-edges incident on it, then Coloring rule 3 or 4 would have been applicable on x_T. If it had 2-edge degree 1, then Coloring rule 6 would have been applicable on x_T. Hence, we can assume that x_T has 2-edge degree exactly 2. Now, if $|E(T)| \geq 2$, Coloring rule 7 becomes applicable on x_T. Hence, we can assume that the tree T is just an edge. Let uv be this edge. Let the 2 2-edges incident on x_T be yx_T and zx_T.*
Case 1: Both yx_T and zx_T are incoming to x_T (see Figure 5.6(a)).
Then the outgoing edge of u in B_1 is clearly 1-edge, say $\overrightarrow{x_T w}$. Assume without loss of generality that its representative is vw. Let $c_1 = s(T)$ and c_2 be the color of uv. If yu

and yv are uncolored, color them with c_1 and c_2 respectively. If zu and zv are uncolored, color both of them with c_2.
Case 2: One of the 2-edges, say yx_T is outgoing from x_T (see Figure 5.6(b)).
Let c_2 be the color of uv and c_3 be the color of representative of any 1-edge incoming on x_T. Note that at least one such 1-edge exists as the degree of x_T is at least 3. If yu and yv are uncolored, color them with c_3 and c_2 respectively. If zu and zv are uncolored, color both of them with c_2.

Path rule 8. *For each tree vertex x_T on which Coloring rule 8 has been applied as above and for each P_{ab} such that Q_{ab} contains at least one of yx_T and zx_T (we say that the path rule is being applied on the pair (x_T, P_{ab})):*
If Q_{ab} contains both yx_T and zx_T then add yu and uz to P_{ab}. If Q_{ab} contains only yx_T and not zx_T then let $y_1 := y$. If Q_{ab} contains only zx_T and not yx_T then let $y_1 := z$. If $a \in V(T)$, let $y_2 := a$, and if $b \in V(T)$ let $y_2 := b$. (Note that both a and b cannot be in T as Q_{ab} contains yx_T or zx_T). If $a, b \notin V(T)$ and only one of yx_T, zx_T is in Q_{ab} then there is an edge $e'' \notin \{yx_T, zx_T\}$ incident on x_T in Q_{ab}; and since yx_T and zx_T are the only 2-edges incident on x_T, the edge e'' is a 1-edge; let y_2 be the endpoint of $(e'')_1$ in T. Add the edge y_1y_2 to P_{ab}.

Lemma 5.35. *Invariant 1 is not violated during Path rule 8.*

Proof. Suppose the invariant is violated. Then there exist edges e and e' in P_{ab} having the same color. We can assume without loss of generality that e was colored during the application of Coloring rule 8 on (x_T, P_{ab}). We added at most 2 edges during the application of Path rule 8 on x_T and if we added 2 edges we have made sure they have distinct colors. Thus e' was not added during the application of Path rule 8 on (x_T, P_{ab}). The colors that are possible for e are c_1, c_2 and c_3 according to Coloring rule 8.
Case I. $c(e) = c(e') = c_2$.
This is not possible since c_2, the color of uv, has not been used to color any other edges so far.
Case II. $c(e) = c(e') = c_1 = s(T)$. This means $e = yu$ and that Case 1 of Coloring rule 8 (Figure 5.6(a)) was applied on x_T. The only coloring rules so far that uses surplus colors are Coloring rules 8, 5, 4, 3, and 2.
Case II.a. e' was colored during Coloring rule 2. This means e' is a 1-edge. Now, the only way e' can have color $s(T)$ is if $e' = vw$. But, in Path rule 8, we add $yu = e$ to P_{ab} only when vw is not in Q_{ab}. Thus we have a contradiction.
Case II.b. e' was colored during Coloring rules 4 or 3. Let T' be the tree on which e' is incident. Since e' was colored with $s(T)$ during Coloring rules 4 or 3, we know that Q_{ab} does not intersect $\mathtt{ST}(x_{T'}, x_T)$ due to Lemmas 5.29 and 5.26. Then Q_{ab} does not contain x_T and hence P_{ab} does not contain e, which is a contradiction.
Case II.c. e' was colored during Coloring rule 5. Then $h(e')$ is a 2-edge in the path from x_T to root of B_1. But then by Path rule 5, we would have added yv instead of $yu = e$, a contradiction.
Case II.d. e' was colored during Coloring rule 8. The only application of Coloring rule 8 that uses $s(T)$ is the application on x_T. But since e' was not colored during this application, we have a contradiction.
Case III. $c(e) = c(e') = c_3$.

This means $e = uy$ and that Case 2 of Coloring rule 8 was applied on x_T.
Let x be the neighbor of x_T such that xx_T is the 1-edge incident on x_T whose representative is colored with c_3. There exist a tree T' such that $s(T') = c_3$. The only coloring rules so far that uses surplus colors are Coloring rules 8, 5, 4, 3, and 2.
Case III.a. e' was colored during Coloring rule 2. This means e' is a 1-edge. Since xx_T is the only 1-edge with color $s(T')$, we have that $e' = xx_T$. Hence xx_T is in Q_{ab}. But if xx_T is in Q_{ab}, we would have added vy and not uy in Path rule 8. This is a contradiction to $e = uy$.
Case III.b. e' was colored during Coloring rules 4 or 3.
Let T'' be such that e' is adjacent on $x_{T''}$. Since P_{ab} contains e' that is incident on $x_{T''}$, we have that Q_{ab} contains $x_{T''}$. Since P_{ab} contains e that is incident on x_T, we have that Q_{ab} contains x_T. This implies that Q_{ab} contains x as x is on the path from x_T to $x_{T''}$. But then we would have added vy and not $uy = e$ to P_{ab} according to Path rule 8, a contradiction.
Case III.c. e' was colored during Coloring rule 5. Then, e' was colored during the application of the rule on x. This implies that Q_{ab} contains x. But then we would have added vy and not $uy = e$ to P_{ab} according to Path rule 8, a contradiction.
Case III.d. e' was colored during Coloring rule 8. Then, e' was colored during the application of the rule on T', a child of x. This implies that Q_{ab} contains x. But then we would have added vy and not $uy = e$ to P_{ab} according to Path rule 8, a contradiction. □

The following Lemma follows from the previous coloring rules.

Lemma 5.36. *Consider an edge e in B_1 that remains uncolored after the application of Rules 1 to 8. Let x_T and v be the endpoints of e (recall that e is a 2-edge and one of the endpoints has to be a tree vertex). Then, both x_T and v have degree exactly 2 in B_1, both edges incident on x_T are 2-edges, and T consists of just a single edge.*

Proof. Suppose $u \in \{v, x_T\}$ had degree not equal to 2 in B_1. First, suppose the degree was greater than 2. Then Coloring rule 8 or 5 would have been applicable on u, and hence u would have been completed. Therefore u has degree 1 in B_1. By Corollary 5.19, every leaf of B_1 is a tree vertex. Hence u is a tree vertex and hence $u = x_T$. But then Coloring rule 6 would have been applicable on x_T, and x_T would have been completed. Thus e is already colored, which is a contradiction. Hence, x_T and v have degree 2 in B_1.

Suppose x_T has only one 2-edge incident in B_1. Then, Coloring rule 6 would have been applied on x_T and x_T would have been completed. Thus, both edges incident on x_T in B_1 are 2-edges. If T contained at least 2 edges, Coloring rule 7 would have been applied on x_T and x_T would have been completed. Hence T contains only 1 edge. □

Coloring rule 9. *For each tree vertex x_T having exactly 1 uncolored 2-edge incident, say e: by Lemma 5.36, the tree T is just a single edge, say e'. Color $(e)_1$ and $(e)_2$ with the color of e'.*

Path rule 9. *For each tree vertex x_T on which Coloring rule 9 has been applied as above and for each P_{ab} such that Q_{ab} contains e (we say that the path rule is being applied on the pair (x_T, P_{ab})):*
If $a \in V(T)$, let $w := a$, and if $b \in V(T)$ let $w := b$. (Note that both a and b cannot

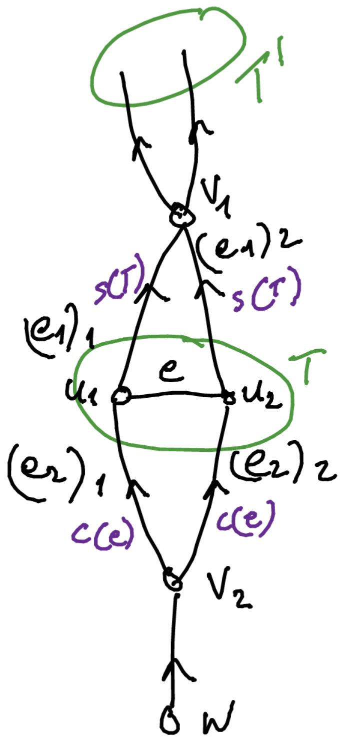


Figure 5.7: Coloring rule 10

be in T as Q_{ab} contains e). If $a, b \notin V(T)$ then there is an edge $e_2 \neq e$ of Q_{ab} incident on x_T; and since e is the only uncolored edge incident on x_T, a representative of the edge e_2 is already in P_{ab}; let w be the endpoint in T of this representative of e_2. By Observation 5.22, there is a path in T that excludes e' from w to one of the foots of e. Let this foot be z. Add the path in T between w and z to P_{ab}. Also add to P_{ab}, the representative of e, having z as its endpoint in T.

Lemma 5.37. *Invariant 1 is not violated during Path rule 9.*

Proof. The edges added to P_{ab} during the application of Path rule 9 to (x_T, P_{ab}) are all having colors from $c(T)$. None of the colors in $c(T)$ were used before anywhere outside T. The path from w to z does not contain e' and the color of the added representative of e is $c(e')$. Thus the edges added are all having distinct colors and these colors were not used in P_{ab} before. □

Lemma 5.38. *Consider a 2-edge e incident on tree vertex x_T that remains uncolored after the application of Rules 1 to 9. Then, x_T has degree exactly 2 in B_1, T contains only one edge, and the other edge incident on x_T is an uncolored 2-edge.*

Proof. By Lemma 5.36 it follows that x_T has degree exactly 2 in B_1, T contains only one edge, and the other edge incident on x_T is a 2-edge. If this other 2-edge is colored, then Coloring rule 9 would have been applied on x_T and x_T would have been completed. □

Coloring rule 10. *For each incomplete tree vertex x_T whose parent's outgoing edge is a 2-edge: (See Figure 5.7 for an Illustration). Let v_1 be the parent of x_T. From Lemma 5.38, it follows that x_T has degree exactly 2 in B_1, has one incoming and one outgoing 2-edge incident on it, both the 2-edges are uncolored, and the tree T is just a single edge. Let e_1 and e_2 respectively be the outgoing and incoming 2-edges of x_T. Let e be the only edge in T. Let v_2 be the other end point of e_2. Let u_1 be the endpoint of $(e_1)_1$ and $(e_2)_1$ in T. Let u_2 be the endpoint of $(e_1)_2$ and $(e_2)_2$ in T. From Lemma 5.36, we know that v_1 and v_2 have degree exactly 2. Let $\overrightarrow{v_1x_{T'}}$ be the outgoing edge from v_1 and $\overrightarrow{wv_2}$ be the unique incoming edge on v_2 in B_1.*
Color $(e_2)_1$ and $(e_2)_2$ with the color of e, and color $(e_1)_1$ and $(e_1)_2$ with $s(T)$.

Path rule 10. *For each tree vertex x_T on which Coloring rule 10 has been applied as above and for each P_{ab} such that Q_{ab} contains e_1 or e_2 (we say that the path rule is being applied on the pair (x_T, P_{ab})):*

Case A. If Q_{ab} contains both e_1 and e_2: Add v_1u_1 and v_1u_2 to P_{ab}.

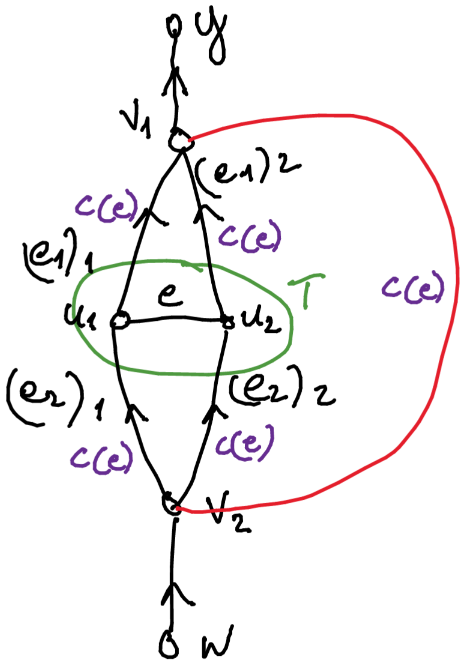
Case B. If Q_{ab} contains exactly one edge among e_1 and e_2: Then either a or b is in $V(T)$. Also both of them cannot be in $V(T)$. Let z be the one among a or b that is in $V(T)$.

Case B.1. If Q_{ab} contains e_1: Add v_1z to P_{ab}.

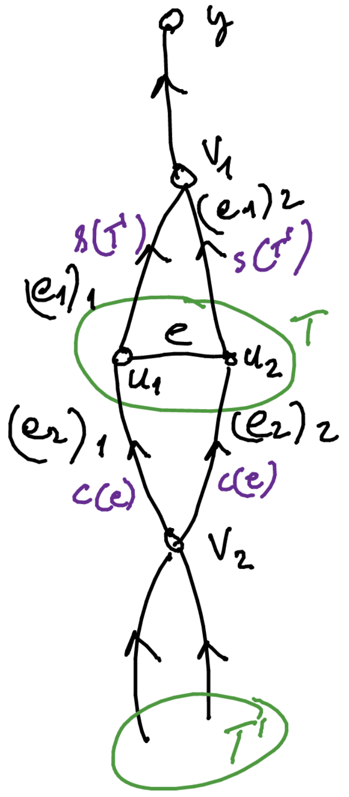
Case B.2. If Q_{ab} contains e_2: Add v_2z to P_{ab}.

Lemma 5.39. *Invariant 1 is not violated during Path rule 10.*

Proof. Suppose for the sake of contradiction that the invariant is violated. Then there exist distinct edges d_1 and d_2 in P_{ab} having the same color. We can assume without loss of generality that d_1 was colored during the application of Coloring rule 10 on x_T. We added at most 2 edges during the application of Path rule 10 on x_T, and in the cases where we added 2 edges, we have made sure that the 2 edges have distinct colors. Thus d_2 was not added during the application of Path rule 10 on x_T and hence was not colored during the application of Coloring rule 10 on x_T. The colors that are possible for d_1 are $s(T)$ and $c(e)$.
Case I. $c(d_1) = c(d_2) = c(e)$.
This is not possible since the color of e has not been used to color any other edges so far.
Case II. $c(d_1) = c(d_2) = s(T)$. This means $h(d_1) = v_1x_T$. The only coloring rules so far that use the surplus colors of trees are Coloring rules 2, 3, 4, 5, 8, and 10. Hence d_2 was colored with $s(T)$ during one of them.
Case II.a. d_2 was colored during Coloring rule 2. This means that d_2 is a 1-edge. The only 1-edge that can be colored with $s(T)$ is either the outgoing edge of x_T or the outgoing edge of the parent of x_T. However, both of them are 2-edges and hence we have a contradiction.
Case II.b. d_2 was colored during Coloring rules 4 or 3. Let T'' be such that d_2 is adjacent on $x_{T''}$. Then, by Lemmas 5.29 and 5.26, we know that Q_{ab} does not intersect $\mathtt{ST}(x_{T''}, x_T)$, in particular Q_{ab} does not contain x_T. This implies P_{ab} does not contain d_1, which is a contradiction.
Case II.c. d_2 was colored during application of Coloring rule 5. Then, d_2 was colored during application of Coloring rule 5 on some ancestor v' of x_T such that there are no other tree vertices in the path from x_T to v'. Then, the only possibility for v' is v_1 as



(a) Coloring rule 11: Case 1. The red color of edge v_1v_2 is to highlight that the edge is not in B_1.



(b) Coloring rule 11: Case 2

the parent of v_1 is a tree vertex. However, we know that v_1 has degree 2 in B_1, and hence Coloring rule 5 could not have been applied on v_1. Thus, we have a contradiction.
Case II.d. d_2 was colored during application of Coloring rule 8. From Coloring Rule 8, this implies that there is a 1-edge with color $s(T)$. The only 1-edge that can be colored with $s(T)$ is either the outgoing edge of x_T or the outgoing edge of the parent of x_T. However, both of them are 2-edges and hence we have a contradiction.
Case II.e. d_2 was colored during Coloring rule 10. Since d_2 was not colored during the application of Coloring rule 10 on x_T, we have that d_2 was colored during the application of Coloring rule 10 on some $x_{T''} \neq x_T$. But then d_2 is not colored with $s(T)$, a contradiction. □

Coloring rule 11. *For each incomplete tree vertex x_T: From Lemma 5.38, it follows that x_T has degree exactly 2 in B_1, has one incoming and one outgoing 2-edge incident on it, both the 2-edges are uncolored, and the tree T is just a single edge. Let e_1 and e_2 respectively be the outgoing and incoming 2-edges of x_T. Let e be the only edge in T. Let v_1 be the other end point of e_1 and v_2 be the other end point of e_2. Let u_1 be the endpoint of $(e_1)_1$ and $(e_2)_1$ in T. Let u_2 be the endpoint of $(e_1)_2$ and $(e_2)_2$ in T. From Lemma 5.36, we know that v_1 and v_2 have degree exactly 2. Let $\overrightarrow{v_1y}$ be the outgoing edge from v_1 and $\overrightarrow{wv_2}$ be the unique incoming edge on v_2 in B_1. Since Coloring rule 10 was not applicable on x_T, we have that v_1y is a 1-edge.*
Case 1. There is an edge between v_1 and v_2 in G. (See Figure 5.8(a) for an illustration).

Color the representatives of e_1 and e_2 with $c(e)$. Color v_1v_2 it with $c(e)$. We say that v_1v_2 is a **shortcut edge**. *Note that shortcut edges are the only colored edges in G that are outside of $g(B)$.*
Case 2. Case 1 does not apply. (See Figure 5.8(b) for an illustration).
We prove in Lemma 5.44 that wv_2 is a 2-edge in this case. Let $T' = f_T(w)$. Color $(e_1)_1$ and $(e_1)_2$ with $s(T')$ and color $(e_2)_1$ and $(e_2)_2$ with color of e.

Path rule 11. *For each tree vertex x_T on which Coloring rule 11 has been applied as above and for each P_{ab} such that Q_{ab} contains e_1 or e_2 (we say that the path rule is being applied on the pair (x_T, P_{ab})):*
Case A. If Q_{ab} contains both e_1 and e_2:
Case A.1. If $v_1v_2 \in E(G)$: Add v_1v_2 to P_{ab}.
Case A.2. If $v_1v_2 \notin E(G)$: Add v_1u_1 and v_2u_1 to P_{ab}.
Case B. If Q_{ab} contains exactly one edge among e_1 and e_2: Then either a or b is in $V(T)$. Also both of them cannot be in $V(T)$. Let z be the one among a or b that is in $V(T)$.
Case B.1. If Q_{ab} contains e_1: Add v_1z to P_{ab}.
Case B.2. If Q_{ab} contains e_2: Add v_2z to P_{ab}.

Lemma 5.40. *Invariant 1 is not violated during Path rule 11.*

Proof. Suppose for the sake of contradiction that the invariant is violated. Then there exist distinct edges d_1 and d_2 in P_{ab} having the same color. We can assume without loss of generality that d_1 was colored during the application of Coloring rule 11 on x_T. We added at most 2 edges during the application of Path rule 11 on x_T, and in the cases where we added 2 edges, we have made sure that the 2 edges have distinct colors. Thus d_2 was not added during the application of Path rule 11 on x_T and hence was not colored during Coloring rule 11 on x_T. The colors that are possible for d_1 are $c(e)$ and $s(T')$.
Case I. $c(d_1) = c(d_2) = c(e)$.
This is not possible since the color of e has not been used to color any other edges so far.
Case II. $c(d_1) = c(d_2) = s(T')$.
This means $h(d_1) = e_1$ and that Case 2 of Coloring rule 11 was applied on x_T. The only coloring rules so far that use the surplus colors of trees are Coloring rules 2, 3, 4, 5, 8, 10, and 11. Hence d_2 was colored with $s(T)$ during one of them.
Case II.a. d_2 was colored during Coloring rule 2.
This means d_2 is a 1-edge and $h(d_2)$ is either $x_{T'}v_2$ or v_2x_T. But since both $x_{T'}v_2$ and v_2x_T are 2-edges (since Case 2 of Coloring rule 11 was applied on x_T), this is not possible.
Case II.b. d_2 was colored during Coloring rules 4 or 3.
Let T'' be the tree such that d_2 is adjacent on $x_{T''}$. By Lemmas 5.29 and 5.26, we know that Q_{ab} does not intersect $\texttt{ST}(x_{T''}, x_{T'})$. Then x_T is not in $\texttt{ST}(x_{T''}, x_{T'})$. This implies $x_{T''}$ is in the path from x_T to $x_{T'}$. But the only vertex in the path from x_T to $x_{T'}$ is v_2, a non-tree vertex. Thus, we have a contradiction.
Case II.c. d_2 was colored during application of Coloring rule 5. Since v_2 has degree 2, $s(T')$ could not have been used in Coloring rule 5. This is a contradiction.
Case II.d. d_2 was colored during application of Coloring rule 8.
Case II.d.1. d_2 was colored during application of Coloring rule 8 Case 1.
Then, d_2 was colored during application of Coloring rule 8 on $x_{T'}$.

But the outgoing edge of $x_{T'}$ in B_1 is a 1-edge. This means that Case 1 of Coloring rule8 could not have been applied on $x_{T'}$, a contradiction.
Case II.d.2. d_2 was colored during application of Coloring rule 8 Case 2.
Then, there is a 1-edge having the color $s(T')$. But there are no 1-edges having the color $s(T')$, as for $x_{T'}$, both the outgoing edge and the outgoing edge of the parent are 2-edges. Hence, we have a contradiction.
Case II.e. d_2 was colored during application of Coloring rule 10.
Since $c(d_2) = s(T')$, from Coloring rule 10 we get that d_2 was colored during the applicaiton of Coloring rule 10 on $x_{T'}$. In Lemma 5.45, we prove that Coloring rule 10 was not applied on $x_{T'}$. Thus, we have a contradiction.
Case II.f. d_2 was colored during application of Coloring rule 11.
Since $c(d_2) = s(T')$, from Coloring rule 11, we get that d_2 was colored during the application of Coloring rule 11 on x_T. Moreover, $h(d_2) = e_1$. Recall that we have $h(d_1) = e_1$ too. Since we picked only one representative of e_1 into P_{ab} by Path rule 11, we have that $d_1 = d_2$. This is a contradiction to the fact that d_1 and d_2 are distinct. □

The following lemma follows from our selection of the skeleton using the configuration vector.

Lemma 5.41. *Any non-tree vertex v having degree 2 in B_1 has no 2-edge to any vertex in H except possibly to its 2 neighbors in B_1.*

Proof. Let z_1 and z_2 be the neighbors of v in B_1 and let z_1 be the parent of v. Suppose for the sake of contradiction that v has a 2-edge in H to a tree vertex $x_{T_1} \in V(H) \setminus \{z_1, z_2\}$. This means vx_{T_1} is not in B too, as x_{T_1} being a tree vertex, is not in L_S. We show that a skeleton with lexicographically higher configuration vector than B exists, thereby giving a contradiction.

Case 1.1. $x_{T_1} \in \mathtt{ST}(v, z_1)$: Let $\overrightarrow{zx_{T_1}}$ be the incoming edge of x_{T_1} in B_1. Let B' be the skeleton obtained by deleting $\overrightarrow{vz_1}$ from B and adding $\overrightarrow{vx_{T_1}}$. Going from B to B', the number of 2-edges is non-decreasing, the degree of x_{T_1} increases, degree of z_1 decreases, and degree of all other vertices remains same. Since z_1 is at a smaller level than x_{T_1} in both B and B', B' has higher configuration vector than B. Thus we have a contradiction to the choice of B.

Case 1.2. $x_{T_1} \notin \mathtt{ST}(v, z_1)$: Then clearly $x_{T_1} \in \mathtt{ST}(v, z_2)$ as v has degree 2 in B_1. Let $\overrightarrow{x_{T_1}z}$ be the outgoing edge of x_{T_1} in B_1. Let B' be the skeleton obtained by deleting $\overrightarrow{x_{T_1}z}$ from B and adding $\overrightarrow{x_{T_1}v}$. Going from B to B', the number of 2-edges is non-decreasing, degree of v increases, degree of z decreases, and degree of all other vertices remains same. Since v is at a smaller level than z in both B and B', B' has higher configuration vector than B. Thus we have a contradiction to the choice of B. □

The following corollary is immediate from the above lemma.

Corollary 5.42. *Let x_T be a vertex on which Coloring Rule 11 is being applied. Let v_1, v_2, y, w be as defined in Coloring Rule 11. The vertex v_1 has no 2-edge in H to any vertex in $V(H)\setminus\{y, x_T\}$. Similarly, v_2 has no 2-edge in H to any vertex in $V(H)\setminus\{w, x_T\}$.*

Lemma 5.43. *Let x_T be a vertex on which Coloring Rule 11 is being applied. Let v_1, v_2, y, w, u_1, u_2 be as defined in Coloring Rule 11. If v_1y is a 1-edge, vertex v_1 has no* *edge in H to any tree vertex except x_T (which also implies that y is a non-tree vertex in this case). Similarly if wv_2 is a* 1*-edge, vertex v_2 has no edge in H to any tree vertex except x_T (which also implies that w is a non-tree vertex in this case).*

Proof. We prove here only the first claim of the lemma, the second follows similarly. Suppose v_1 has an edge in H to a tree vertex $x_{T_1} \neq x_T$. Then $v_1x_{T_1}$ is not a 2-edge due to Corollary 5.42. Thus $v_1x_{T_1}$ is a 1-edge. Let $F' = (F \setminus \{u_1\}) \cup \{v_1\}$. We prove that $\mathcal{F}' = G[F']$ is a forest with fewer number of trees than $\mathcal{F}$, which is a contradiction to the choice of $\mathcal{F}$.

First, we prove that $\mathcal{F}'$ is indeed a forest. Suppose for the sake of contradiction that there is a cycle C in $\mathcal{F}'$. The cycle C has to contain v_1 because otherwise C is also a cycle in $\mathcal{F}$. The cycle C can intersect at most one tree in $\mathcal{T}$ as there are no edges across the trees. Let this tree be T_2. Then v_1 should have 2 edges to T_2 in C in order to complete the cycle. We know that v_1 does not have 2-edges to any tree in $\mathcal{T} \setminus \{T\}$ due to Corollary 5.42. Hence $T_2 = T$. But, since $|V(T) \cap F'| = 1$, v can only have one edge in C to T. Thus, we have a contradiction. Hence, $\mathcal{F}'$ is indeed a forest.

Now, we show that the number of trees in $\mathcal{F}'$ is smaller than that of $\mathcal{F}$. Let $\mathcal{T}_1$ be the set of trees in $\mathcal{T} \setminus \{T\}$ that have an edge from v_1 in G. Clearly $T_1 \in \mathcal{T}_1$ and hence $|\mathcal{T}_1| \geq 1$. The vertex set $\{u_2, v_1\} \cup \bigcup_{T'' \in \mathcal{T}_1} V(T'')$ induces a tree in G. Hence the number of trees in $\mathcal{F}'$ is at most $|\mathcal{T} \setminus (\mathcal{T}_1 \cup \{T\})| + 1 \leq |\mathcal{T}| - |\mathcal{T}_1| \leq |\mathcal{T}| - 1$. □

Lemma 5.44. *Let x_T be a vertex on which Coloring Rule 11 is being applied and suppose the precondition of Case* 1 *of the rule is not satisfied. Let v_1, v_2, w be as defined in Coloring Rule 11. Then, wv_2 is a* 2*-edge.*

Proof. Suppose wv_2 is a 1-edge. Let $F' = (F \setminus \{u_1\} \cup \{v_1, v_2\})$. Let $\mathcal{F}' = G[F']$. We will show that $\mathcal{F}'$ is a forest. Then since $|F'| > |F|$, we have that $\mathcal{F}$ is not a maximum induced forest, a contradiction.

Suppose for the sake of contradiction that there is a cycle C in $\mathcal{F}'$.

Case 1. C intersects more than 2 trees of $\mathcal{T}$: Since there are only 2 vertices in F' that are not in any tree in $\mathcal{T}$, there have to be 2 vertices in 2 different trees of $\mathcal{T}$ that are adjacent in C. This is a contradiction as the trees are connected components of $\mathcal{F}$ and therefore have no edges between them in G.

Case 2. C intersects exactly 2 trees of $\mathcal{T}$: Let T_1 and T_2 be the trees that C interesects. By Lemma 5.43, we have that v_1 and v_2 does not have edge to any tree vertex in B_1 except x_T. Hence at least one of T_1 and T_2 have no edges to both v_1 and v_2. This means that there should be an edge in C between T_1 and T_2. This is a contradiction as T_1 and T_2 are connected components of $\mathcal{F}$ and therefore have no edges between them in G.

Case 3. C intersects exactly 1 tree of $\mathcal{T}$: Let T_1 be the tree that C interesects. By Lemma 5.43, we have that v_1 and v_2 does not have edge to any tree vertex in B_1 except x_T. Hence $T_1 = T$. Since only one vertex of T is in F', v_1 and v_2 both can have at most one edge to T in C. This means v_1 and v_2 should be adjacent in C to complete the cycle. However, since the precondition of Case 1 of Rule 11 is not satisfied, v_1 and v_2 are not adjacent in G. Thus we have a contradiction. □

Lemma 5.45. *Let x_T be a vertex on which Coloring Rule 11 is being applied and suppose the precondition of Case* 1 *of the rule is not satisfied. Let v_1, v_2, T' be as defined in Coloring Rule 11. Then, $x_{T'}$ is not a vertex on which Coloring rule 10 was applied.*

Proof. Suppose for the sake of contradiction that Coloring rule 10 was applied on $x_{T'}$. Then $x_{T'}$ has degree 2 in B_1 and T' consists of a single edge. Let this edge be u_3u_4. Observe that the representatives of the outgoing edge of $x_{T'}$ are u_3v_2 and u_4v_2. Let $F' = (F \setminus \{u_1, u_3\} \cup \{v_1, v_2\})$. Let $\mathcal{F}' = G[F']$. Note that $|F'| = |F|$. We will show that $\mathcal{F}'$ is a forest with fewer number of trees than $\mathcal{F}$, thereby showing a contradiction to the choice of $\mathcal{F}$.

First, we show that $\mathcal{F}'$ is indeed a forest. Suppose for the sake of contradiction that there is a cycle C in $\mathcal{F}'$.

Case 1. C intersects more than 2 trees of $\mathcal{T}$: Since there are only 2 vertices in F' that are not in any tree in $\mathcal{T}$, there have to be 2 vertices in 2 different trees of $\mathcal{T}$ that are adjacent in C. This is a contradiction as the trees are connected components of $\mathcal{F}$ and therefore have no edges between them in G.

Case 2. C intersects exactly 2 trees of $\mathcal{T}$: Let T_1 and T_2 be the trees that C interesects. By Lemma 5.43, we have that v_1 does not have edge to any tree vertex in B_1 except x_T. Hence at least one of T_1 and T_2 have no edges to v_1. This means that there should be an edge in C between T_1 and T_2 to complete the cycle. This is a contradiction as T_1 and T_2 are connected components of $\mathcal{F}$ and therefore have no edges between them in G.

Case 3. C intersects exactly 1 tree of $\mathcal{T}$: Let T_1 be the tree that C interesects. Since the precondition of Case 1 of Rule 11 is not satisfied, v_1 and v_2 are not adjacent in G. Hence, only one of them is in C.

Case 3.1. v_1 is in C: Then, v_1 should have 2 edges in C to T_1 in order to complete the cycle. By Lemma 5.43, we have that v_1 does not have an edge in H to any tree vertex in B_1 except x_T. Hence $T_1 = T$. Since, only one vertex of T is in F', v_1 can have at most one edge to T in C. Thus we have a contradiction.

Case 3.2. v_2 is in C: Then, v_2 should have 2 edges in C to T_1 in order to complete the cycle. By Corollary 5.42, we have that v_2 does not have an edge in H to any tree vertex in B_1 except x_T and $x_{T'}$. Hence $T_1 \in \{T, T'\}$. However, since only one vertex of each T and T' is in F', v_1 can have at most one edge to each of T and T' in C. Hence, v_1 has only one edge to T_1. Thus, we have a contradiction.

Thus, we have proved that $\mathcal{F}'$ is indeed a forest. Now, we prove that $\mathcal{F}'$ has fewer number of trees than $\mathcal{F}$, which concludes the proof. Let $\mathcal{T}_1$ be the set of trees in $\mathcal{T} \setminus \{T, T'\}$ that have an edge from v_1 or v_2 in G. The vertex set $\{v_1, u_2, v_2, u_4\} \cup \bigcup_{T'' \in \mathcal{T}_1} V(T'')$ induces a tree in G. Hence the number of trees in $\mathcal{F}'$ is at most $|\mathcal{T} \setminus (\mathcal{T}_1 \cup \{T, T'\})| + 1 \leq |\mathcal{T}| - |\mathcal{T}_1| - 1 \leq |\mathcal{T}| - 1$. □

Now, we have colored the representatives of all edges in B_1. We have also colored some additional edges called shortcut edges. We now show that the vertices in B_1 are rainbow connected through these colored edges.

Lemma 5.46. *For any pair of vertices $v_1, v_2 \in V(G) \setminus L_S$, P_{ab} is a rainbow path between v_1 and v_2 in G and uses only colors in $[f]$.*

Proof. Clearly, after Coloring Rule 11 has been exhaustively applied, there are no incomplete vertices. This means there are no uncolored 2-edges. Also, Coloring Rule 2 colors all 1-edges. Thus, each edge in B_1, and hence their representatives in G, have been colored.

Whenever an edge in B_1 is colored by a coloring rule and if it is in Q_{ab}, we have added exactly one of its representatives to P_{ab} in the proceeding path rule, except in

Case A.1 in Path rule 11 where we have added a shortcut edge instead. In the case when a shortcut edge is added, the shortcut edge shortcuts the two consecutive edges in Q_{ab} whose representatives were not added to P_{ab} and hence the path is not broken. Also, whenever a tree T has 2 edges of P_{ab} incident on it, we have added the path between the endpoints of the edges in the tree to P_{ab}. And, whenever a tree T with $a \in V(T)$ has 1 edge of P_{ab} incident on it, we have added the path between the endpoints of the edge and a in the tree to P_{ab}. Similarly, whenever a tree T with $b \in V(T)$ has 1 edge of P_{ab} incident on it, we have added the path between the endpoints of the edge and b in the tree to P_{ab}. If there is a tree T with $a, b \in V(T)$ we added the path between the endpoints of a and b in the tree to P_{ab} during Path rule 1. It follows that P_{ab} is indeed a path between a and b in G. Since Invariant 1 holds, we know that P_{ab} is a rainbow path. Since we have used only the colors from 1 to f so far, the lemma follows. $\square$

So, now we only need to worry about how to rainbow connect vertices in L_S between themselves and to the other vertices. For this, we give the following coloring rule.

Coloring rule 12. *For each $v \in L_S$: let e be the unique 2-edge incident on v (which exist due to Lemma 5.17); color $(e)_1$ with g_1 and $(e)_2$ with g_2. (Recall that g_1 and g_2 are the global surplus colors $f+1$ and $f+2$ respectively).*

Now, we complete the proof of Theorem 5.3. Consider any pair of vertices $a_1, a_2 \in V(G)$. If $a_1 \in L_S$, let e_1 be the edge incident on a_1 that is colored with g_1, and let a be the other end of e_1. If $a_1 \notin L_S$, let $a = a_1$. If $a_2 \in L_S$, let e_2 be the edge incident on a_2 that is colored with g_2, and let b be the other end of e_2. If $a_2 \notin L_S$, let $b = a_2$. We know there is a rainbow path P_{ab} from a to b that uses only colors in $[f]$ due to Lemma 5.46. We define path P as follows. If $a_1, a_2 \in L_S$, then $P := a_1 a P_{ab} b a_2$. If $a_1 \in L_S$ but $a_2 \notin L_S$, then $P := a_1 a P_{ab}$. If $a_2 \in L_S$ but $a_1 \notin L_S$, then $P := P_{ab} b a_2$. If$a_1, a_2 \notin L_S$, then $P := P_{ab}$. It is clear from the construction that P is a path between a_1 and a_2. Since edge $a_1 a$ is colored with $g_1 = f+1$, edge $b a_2$ is colored with $g_2 = f+2$, and path P_{ab} uses only colors in $[f]$, the path P is indeed a rainbow path.

5.3 Bounds on VSRC and SRC

We show several upper and lower bounds on $\mathsf{vsrc}(G)$, both for general graphs and for graphs G that belong to a specific graph class. Crucial in our analysis are the connections between very strong rainbow colorings and decompositions of the input graph into cliques. The graph $\hat{G}$ used in the following lemma (defined in the preliminaries in Section 5.1.5) is important for our hardness reductions.

Lemma 5.47. *Let G be any graph. Then*

(1) $\mathsf{src}(G) \leq \mathsf{vsrc}(G) \leq \mathsf{cp}(G)(\mathsf{cp}(G)+1)/2.$

(2) $\mathsf{src}(\hat{G}) \leq \mathsf{vsrc}(\hat{G}) \leq \mathsf{cp}(G)(\mathsf{cp}(G)+1)/2.$

Proof. Let $\mathcal{C} = C_1, \ldots, C_r$ be the set of cliques in an optimal clique partition of G; that is, $r = \mathsf{cp}(G)$. For a vertex v, let $c(v)$ denote the clique in $\mathcal{C}$ that contains v. We define the set of colors as $\binom{\mathcal{C}}{1} \cup \binom{\mathcal{C}}{2}$, the set of subsets of $\mathcal{C}$ of size 1 or 2. Note that the size of this set is at most $\mathsf{cp}(G)(\mathsf{cp}(G)+1)/2$. We color an edge $uv \in E(G)$ by $\{c(u), c(v)\}$. It only remains to show that this edge coloring is indeed a very strong rainbow coloring of G.

For the sake of contradiction, suppose that the edge coloring is not a very strong rainbow coloring of G. Then there exist two vertices $s, t \in V(G)$, a shortest path P between s and t, and two distinct edges $uv, wx \in E(P)$ that received the same color. If $c(u) = c(v)$, then the color of uv is $\{c(u)\}$. Since wx has the same color, we get that $c(w) = c(x) = c(u)$. This implies that P uses two edges within the same clique. Then P can be shortcut, contradicting that P is a shortest path between s and t. Hence, $c(u) \neq c(v)$, implying also that $c(w) \neq c(x)$. Since colors of uv and wx are the same, we can without loss of generality assume now that $c(u) = c(w)$ and $c(v) = c(x)$. (Note that it is possible that either $u = w$ or $v = x$ but not both at the same time). Then either the edge uw or the edge vx will shortcut P, a contradiction.

To see the second part of the lemma, color each edge $\hat{u}v$ incident on the universal vertex $\hat{u}$ in $\hat{G}$ by $c(v)$, in addition to the above coloring of the edges of G. Suppose this was not a very strong rainbow coloring of $\hat{G}$. Then there exists vertices u, v such that $u\hat{u}v$ is a shortest path and $u\hat{u}$ and $v\hat{u}$ are colored the same. But then u and v are in the same clique C_i in $\mathcal{C}$. But then uv can shortcut $u\hat{u}v$, a contradiction. □

The following lemma is more consequential for our upper bounds.

Lemma 5.48. *Let G be any graph. Then $\mathsf{vsrc}(G) \leq \mathsf{is}(G) = \mathsf{ecc}(G)$.*

Proof. Let $\mathcal{U} = \{x_1, x_2, \ldots, x_n\}$ be a universe and $\mathcal{F} = \{S_1, S_2, \ldots, S_m\}$ be a family of subsets of $\mathcal{U}$ such that, G is the intersection graph of $\mathcal{F}$ and $|\mathcal{U}| = \mathsf{is}(G)$. Let v_i be the vertex of G corresponding to the set S_i. We take $\mathcal{U} = \{x_1, x_2, \ldots, x_n\}$ as the set of colors, and color an edge between vertices v_i and v_j with any $x \in S_i \cap S_j$. (Note that this intersection is non-empty because of the presence of the edge v_iv_j). Suppose for the sake of contradiction that this is not a very strong rainbow coloring of G. Then there exist two vertices $s, t \in V(G)$, a shortest path P between s and t, and two edges v_iv_j and v_av_b in P that received the same color. Let x be this color. By the construction of the coloring, we have that $x \in S_i \cap S_j \cap S_a \cap S_b$. Hence, $\{v_i, v_j, v_a, v_b\}$ induce a clique in G. But then the path P can be shortcut, a contradiction. □

We remark that a similar result to the above lemma for the case of $\mathsf{src}(G)$, was proved independently by Lauri [102, Prop. 5.3].

Corollary 5.49. *Let G be any graph. Then $\mathsf{vsrc}(G) \leq \min\{\lfloor |V(G)|^2/4 \rfloor, |E(G)|\}$.*

Proof. Directly from $\mathsf{ecc}(G) \leq \min\{\lfloor |V(G)|^2/4 \rfloor, |E(G)|\}$ for any graph [61]. □

Corollary 5.50. *Let G be any graph.*

(1) If G is chordal, then $\mathsf{src}(G) \leq \mathsf{vsrc}(G) \leq |V(G)| - \chi(G) + 1$.

(2) If G is circular-arc, then $\mathsf{src}(G) \leq \mathsf{vsrc}(G) \leq |V(G)|$.

(3) $\mathsf{src}(\mathcal{L}(G)) \leq \mathsf{vsrc}(\mathcal{L}(G)) \leq |V(G)|$, where $\mathcal{L}(G)$ is the line graph of G.

These bounds are (almost) tight in general.

Proof. In each of the three cases, we express the graph as an intersection graph over a suitable universe. Then, Lemma 5.48 implies that the size of the universe is an upper bound on the vsrc of the graph. The required result directly follows in each case.

Every chordal graph is the intersection graph of subtrees of a tree [72]. It is also known that the number of vertices of this tree only needs to be at most $|V(G)|-\omega(G)+1$. (For completeness, we provide a proof of this in Lemma 5.51 below). Since $\omega(G) = \chi(G)$ for chordal graphs, the required statement follows.

For a circular arc graph G, consider any set of arcs whose intersection graph is G. We now construct a different intersection representation. Take the set of second (considering a clockwise ordering of points) endpoints of all arcs as the universe $\mathcal{U}$. Take $S_i \subseteq \mathcal{U}$ as the elements of $\mathcal{U}$ contained in the i-th arc. It is easy to see that G is the intersection graph of $\mathcal{F} = \left\{S_1, S_2, \ldots, S_{|E(G)|}\right\}$.

Finally, consider $\mathcal{L}(G)$. We construct an intersection representation with universe $V(G)$. For each $uv \in E(G)$, let $S_{uv} = \{u, v\}$. Then $\mathcal{L}(G)$ is the intersection graph of $\mathcal{F} = \{S_e : e \in E(G)\}$.

The (almost) tightness for all the three cases follow by taking G as a path. For a path G, we have $\mathsf{vsrc}(G) = |V(G)| - 1$ and $\mathsf{vsrc}(\mathcal{L}(G)) = |V(G)| - 2$. Paths are both chordal and circular-arc. □

We now prove Lemma 5.51 used in the proof of Corollary 5.50. We remark that this lemma is a known fact and the proof is only given for the sake of completeness.

Lemma 5.51. *Let G be any chordal graph. Then G can be represented as the intersection graph of the subtrees of a tree on at most $n - \omega(G) + 1$ vertices.*

Proof. Let $n = |V(G)|$. Since G is a chordal graph, it has a perfect elimination order [75]. In fact, the lexicographic search algorithm that construct a perfect elimination order implies the existence of such an order $v_1, v_2, \ldots, v_n$ such that $v_n, v_{n-1} \ldots, v_{n-\omega(G)+1}$ forms a maximum clique. For any $i = 1, \ldots, n$, let $G_i = G[v_n, v_{n-1}, \ldots, v_i]$.

We claim that for $n - \omega(G) + 1 \geq i \geq 1$, G_i can be represented as the intersection graph of subtrees $S_n, S_{n-1}, \ldots, S_i$ of a tree T_i on at most $n - i - \omega(G) + 2$ vertices, and there exists a bijection f_i from vertices of T_i to maximal cliques of G_i such that for each $n \geq k \geq i$, $S_k = T_i[u \in V(T_i) : v_k \in f_i(u)]$.

We prove the claim by downwards induction on i. The base case is when $i = n - \omega(G) + 1$. In this case, the statement follows by taking T_i as a single vertex u and $f_i(u)$ as the clique $\{v_n, v_{n-1}, \ldots, v_i\}$.

Now, assuming the statement is true for i, we prove it for $i-1$. Since v_{i-1} is simplicial in G_{i-1}, $N_{G_{i-1}}(v_{i-1})$ is a subset of some maximal clique C of G_j. Let t be the vertex in T_j such that $f_j(t) = C$. If all vertices in C are adjacent to v_{i-1} in G_{i-1}, then we take $T_{i-1} = T_i$, $f_{i-1}(t) = f_i(t) \cup \{v_{i-1}\}$, and $f_{i-1}(t') = f_i(t')$ for all $t' \in V(T_{j+1})$ such that $t' \neq t$. It is easy to see that the statement follows in this case. Now suppose that not all vertices in C are adjacent to v_{i-1} in G_{i-1}. Then we take $V(T_{i-1}) = V(T_j) \cup \{u\}$ and $E(T_{i-1}) = E(T_i) \cup \{u, t\}$ where u is a new vertex introduced with $f_{i-1}(u) = N_{G_{i-1}}[v_{i-1}]$. For all $u' \in V(T_{i-1})$ such that $u' \neq u$, we take $f_{i-1}(u') = f_i(u)$. The statement follows from this construction.

By taking $i = 1$ in the statement of the claim, it follows that G can be represented as the intersection graph of subtrees of a tree with at most $n - \omega(G) + 1$ vertices. □

Corollary 5.52. *Let G be any k-perfectly orientable graph. Then,* $\mathsf{src}(G) \leq \mathsf{vsrc}(G) \leq k|V(G)|$.

Proof. Consider any orientation of the edges of G such that the outgoing neighbors of each vertex v can be partitioned into k or fewer cliques. Let $C(v)$ denote these set of cliques for each vertex v. Let $C'(v) := \{S \cup \{v\} : S \in C(v)\}$. Note that $C'(v)$ is also a set of cliques. Observe that $\bigcup_{v \in V(G)} C'(v)$ is an edge clique cover of G because, every edge is outgoing from some vertex v, and will thus be covered by a clique in $C'(v)$. Hence, $\mathsf{vsrc}(G) \leq \mathsf{ecc}(G) \leq k|V(G)|$. □

Since any k-perfectly groupable graph is also k-perfectly orientable, the above bound also applies to k-perfectly groupable graphs. In this context, we prove an interesting converse of the above bound.

Lemma 5.53. *Let G be any graph. If* $\mathsf{vsrc}(G) \leq k$*, then G is k-perfectly groupable.*

Proof. Consider an optimal very strong rainbow coloring μ of G. Consider an arbitrary vertex v of G and let c be any color used in μ. Define the set $Q(c) = \{u \in N(v) : \mu(vu) = c\}$. Suppose there exist two non-adjacent vertices u, w in $Q(c)$. Then uvw is a shortest path between u and w, and thus uv and vw cannot have the same color, a contradiction to the definition of Q. Hence, for each color c used in μ, $Q(c)$ is a clique. Since the number of colors is at most k, the edges incident on v can be covered with at most k cliques. Hence, G is k-perfectly groupable. □

5.4 Hardness Results for VSRC

The hardness results lean heavily on the combinatorial bounds of the previous section. We will also need the following bound, which strengthens Lemma 5.47.

Lemma 5.54. *Let G be any graph. If* $\mathsf{cp}(G) \leq 3$*, then* $\mathsf{vsrc}(\hat{G}) \leq 3$.

Proof. Let C_1, C_2, and C_3 be three cliques into which $V(G)$ can be partitioned. We will color $\hat{G}$ with three colors, say c_1, c_2, and c_3, as follows: For each edge with both endpoints in C_i for $1 \leq i \leq 3$, color it with c_i. For each edge vw with $v \in C_i$, $w \in C_j$ such that $1 \leq i < j \leq 3$, color it with c_k, where $k \in \{1,2,3\} \setminus \{i,j\}$. Finally, for each edge $\hat{u}v$ with $v \in C_i$ for $1 \leq i \leq 3$, color it with c_i.

Suppose, this is not a very strong rainbow coloring of $\hat{G}$. Then, since the diameter of $\hat{G}$ is at most 2, there exists a shortest path xyz with xy and yz having the same color. However, if xy and yz have the same color, by the construction of the coloring, at least two of x, y and z are in the same C_i for $1 \leq i \leq 3$ and the third one is either $\hat{u}$ or in C_i itself. Then, we can shortcut xyz by xz, a contradiction. Hence $\mathsf{vsrc}(\hat{G}) \leq 3$. □

We will also need the following Corollary of Lemma 5.53.

Corollary 5.55. $\mathsf{vsrc}(\hat{G}) \geq \mathsf{cp}(G)$.

Proof. If $\mathsf{vsrc}(\hat{G}) \leq k$, then $\hat{G}$ is k-perfectly groupable by Lemma 5.53. In particular, the neighborhood of $\hat{u}$ (the universal vertex in $\hat{G}$) can be partitioned into at most k cliques. Since the graph induced by the neighborhood of $\hat{u}$ in $\hat{G}$ is nothing but the graph G, we have that $\mathsf{cp}(G) \leq k$. □

We will now prove Theorem 5.4. We state the theorem here again for convenience.

Theorem 5.4. 3-VSRC *is NP-complete. Moreover, there is no polynomial-time algorithm that approximates* $\mathsf{vsrc}(G)$ *within a factor* $|V(G)|^{1-\varepsilon}$ *for any* $\varepsilon > 0$*, unless P=NP.*

Proof. We first prove that 3-VSRC is NP-complete. We will give a reduction from the NP-hard 3-COLORING problem [70]. Let G be an instance of 3-COLORING. Let H be the complement of G. We claim that $\mathsf{vsrc}(\hat{H}) = 3$ if and only if G is 3-colorable. Indeed, if $\mathsf{vsrc}(\hat{H}) \leq 3$, then $\mathsf{cp}(H) \leq 3$ by Corollary 5.55, implying that G is 3-colorable. For the other direction, note that if G is 3-colorable, then $\mathsf{cp}(H) \leq 3$, and by Lemma 5.54, $\mathsf{vsrc}(\hat{H}) \leq 3$.

To prove the hardness of approximation, we recall that, for every $\varepsilon > 0$, there exists a polynomial-time algorithm that takes a SAT formula ψ as input and produces a graph G as output such that: if ψ is not satisfiable then $\mathsf{cp}(G) \geq |V(G)|^{1-\varepsilon}$, and if ψ is satisfiable, then $\mathsf{cp}(G) \leq |V(G)|^{\varepsilon}$ [164, Proof of Theorem 2]. Now,

$$\begin{aligned}
\psi \text{ not satisfiable} &\Rightarrow \mathsf{cp}(G) \geq (|V(G)|)^{1-\varepsilon} \\
&\Rightarrow \mathsf{vsrc}(\hat{G}) \geq (|V(G)|)^{1-\varepsilon} && \text{(as } \mathsf{vsrc}(\hat{G}) \geq \mathsf{cp}(G) \text{ by Corollary 5.55)} \\
&\Rightarrow \mathsf{vsrc}(\hat{G}) \geq (|V(\hat{G})| - 1)^{1-\varepsilon} \\
\psi \text{ satisfiable} &\Rightarrow \mathsf{cp}(G) \leq (|V(G)|)^{\varepsilon} \\
&\Rightarrow \mathsf{vsrc}(\hat{G}) \leq (|V(G)|)^{2\varepsilon} && \text{(as } \mathsf{vsrc}(G) \leq \mathsf{cp}(G)^2 \text{ by Lemma 5.47)} \\
&\Rightarrow \mathsf{vsrc}(\hat{G}) \leq (|V(\hat{G})| - 1)^{2\varepsilon}
\end{aligned}$$

The required inapproximability result follows by rescaling ε. □

5.5 Algorithm for VSRC in Cactus Graphs

Throughout this section, let G be the input cactus graph. We first prove several structural properties of cactus graphs, before presenting the algorithm.

5.5.1 Definitions and Structural Properties of Cactus Graphs

We make several structural observations related to cycles. See Figure 5.9 for an illustration of the concepts introduced in this section. For a vertex v and a cycle C containing v, we define $\mathsf{S}(v, C)$ as the vertices of G that are reachable from v without using any edge of C.

Observation 5.56. *For any cycle C in G, $\{\mathsf{S}(v, C) : v \in V(C)\}$ is a partition of $V(G)$.*

From Observation 5.56, we have that for any fixed $u \in V(G)$ and any fixed cycle C of G, there exists a unique vertex $v \in V(C)$ such that $u \in \mathsf{S}(v, C)$. We denote that unique vertex v by $\mathsf{g}(u, C)$.

Observation 5.57. *Let $u \in V(G)$ and let C be a cycle in G. Let $w \in V(C)$ and let $x_1x_2\ldots x_r$ be a path from u to w where $x_1 = u$ and $x_r = w$. Let i^* be the smallest i such that $x_i \in V(C)$. Then, $x_{i^*} = \mathsf{g}(u, C)$. In simpler words, any path from u to any vertex in C enters C through $\mathsf{g}(u, C)$.*

Observation 5.58. *For any cycle C in G and for any $uv \in E(G) \setminus E(C)$, $\mathsf{g}(u, C) = \mathsf{g}(v, C)$.*

We now focus on even cycles. For an edge uv in an even cycle C, we define its *opposite edge*, denoted by $\mathsf{eopp}(uv)$, as the unique edge $xy \in E(C)$ such that $\mathsf{dist}(u, x) = \mathsf{dist}(v, y)$. Note that $\mathsf{eopp}(\mathsf{eopp}(e)) = e$. Call the pair of edges e and $\mathsf{eopp}(e)$ an *opposite pair.* Each even cycle C has exactly $\frac{|C|}{2}$ opposite pairs.

Lemma 5.59. *Let C be an even cycle. For any vertex $x \in V(G)$ and edge $uv \in E(C)$, either there is a shortest path between x and u that contains uv or there is a shortest path between x and v that contains uv.*

Proof. Let $w = \mathsf{g}(x, C)$. Then, w cannot be equidistant from u and v, because otherwise C is an odd cycle. Suppose that $\mathsf{dist}(w, u) < \mathsf{dist}(w, v)$. Then a shortest path from w to u appended with the edge uv gives a shortest path between w and v. Now, due to Observation 5.57, if we append a shortest path between x and w with a shortest path between w and v, we get a shortest path between x and v. Thus there is a shortest path between x and v that contains uv. If $\mathsf{dist}(w, u) > \mathsf{dist}(w, v)$, then we get the other conclusion of the lemma. □

We now move on to odd cycles. For any edge e in an odd cycle C, there is a unique vertex in C that is equidistant from both endpoints of e. We call this vertex the *opposite vertex* of e and denote it as $\mathsf{vopp}(e)$. We call $\mathsf{OS}(e) := G[\mathsf{S}(\mathsf{vopp}(e), C)]$ the *opposite subgraph* of e. See Figure 5.9 for an illustration.

Lemma 5.60. *Let C be an odd cycle and $uv \in E(C)$. For any vertex $x \in V(G) \setminus V(\mathsf{OS}(uv))$, either there is a shortest path between x and u that contains uv or there is a shortest path between x and v that contains uv.*

Proof. Let $w = \mathsf{g}(x, C)$. Since $x \notin V(\mathsf{OS}(uv))$, $w \neq \mathsf{vopp}(uv)$. Hence, w cannot be equidistant from u and v. So, the same arguments as in Lemma 5.59 complete the proof. □

Lemma 5.61. *Let e be any edge in an odd cycle of G for which $\mathsf{vopp}(e)$ has degree more than 2. Then $\mathsf{OS}(e)$ contains either a bridge, or an even cycle, or an edge e' in an odd cycle such that $\mathsf{vopp}(e')$ has degree 2.*

Proof. Suppose this is not the case. We define a sequence $e_1, e_2, \ldots$ of edges by the following procedure. Let $e_1 = e$. Given e_i, we define e_{i+1} as follows. By assumption and the definition of cactus graphs, e_i is contained in an odd cycle, which we denote by C_i, and $\mathsf{vopp}(e_i)$ has degree more than 2. Choose e_{i+1} as any edge incident on $\mathsf{vopp}(e_i)$ that is not in C_i. However, observe that $\mathsf{OS}(e_{i+1}) \subset \mathsf{OS}(e_i)$ by the choice of e_{i+1}. Hence, this is an infinite sequence, which contradicts the finiteness of $E(G)$. □

5.5.2 Properties of Very Strong Rainbow Colorings of Cactus Graphs

We initially partition the edges of G into three sets: E_{bridge}, E_{even}, and E_{odd}. The set E_{bridge} consists of those edges that are not in any cycle. In other words, E_{bridge} is the set of bridges in G. By definition of cactus graphs, each of the remaining edges is part of

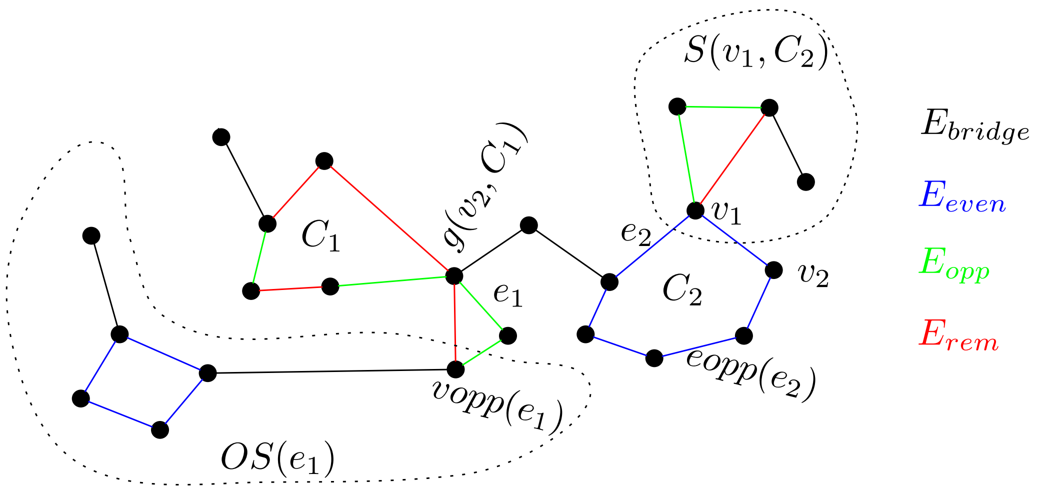


Figure 5.9: An example of a cactus graph and related definitions.

exactly one cycle. We define E_{even} as the set of all edges that belong to an even cycle, and E_{odd} as the set of all edges that belong to an odd cycle. Note that E_{bridge}, E_{even}, and E_{odd} indeed induce a partition of $E(G)$. We then partition E_{odd} into two sets: E_{opp} and E_{rem}. An edge $e \in E_{\mathsf{odd}}$ is in E_{opp} if $\mathsf{vopp}(e)$ is not a degree-2 vertex and in E_{rem} otherwise. See Figure 5.9. We analyze each of these sets in turn, and argue how an optimal VSRC might color them.

Two edges e_1 and e_2 are called *conflicting* if there is a shortest path in the graph which contains both e_1 and e_2. Two conflicting edges must have different colors in any VSRC. We now exhibit several classes of conflicting pairs of edges.

Lemma 5.62. [2] *Any VSRC of G colors the edges of E_{bridge} with distinct colors.*

Proof. Consider $uv, xy \in E_{\mathsf{bridge}}$. We prove that uv and xy are conflicting, i.e. there is a shortest path in G which contains both uv and xy. Since uv is a bridge, we can assume without loss of generality that any path between u and y uses the edge uv. Similarly, since xy is a bridge, we can assume without loss of generality that any path between y and u uses the edge xy. Hence, the shortest path from u to y uses both uv and xy. Hence, uv and xy are conflicting. □

Lemma 5.63. *Let $e_1 \in E_{\mathsf{bridge}}$ and $e_2 \in E_{\mathsf{even}}$. Then any VSRC of G colors e_1 and e_2 with different colors.*

Proof. Let C be the cycle containing e_2. Let $e_1 = xy$ and $e_2 = uv$. Since xy is a bridge, we can assume w.l.o.g. that any path from x to any vertex in C contains xy. Due to Lemma 5.59, we can assume w.l.o.g. that there is a shortest path from x to v that contains uv. Thus we have a shortest path which contains both uv and xy, which means that uv and xy are conflicting. □

Observation 5.64. *Let e_1 and e_2 be edges in an even cycle C of G such that $e_1 \neq$ $\mathsf{eopp}(e_2)$. Then any VSRC of G colors e_1 and e_2 with different colors.*

[2]This lemma holds for any graph, not necessarily cactus

Lemma 5.65. *Let e_1 and e_2 be edges in two different even cycles C_1 and C_2 of G. Then any VSRC of G colors uv and xy with different colors.*

Proof. Let $e_1 = uv$ and $e_2 = xy$. Let $z = \mathsf{g}(u, C_2)$ and $w = \mathsf{g}(x, C_1)$. By Observation 5.58, $\mathsf{g}(v, C_2) = z$ and $\mathsf{g}(y, C_1) = w$. Due to Lemma 5.59, we can assume w.l.o.g. that there is a shortest path P_1 between z and x containing xy and that there is a shortest path P_2 between w and u containing uv. Let P_3 be a shortest path between w and z. Then $P_1 \cup P_3 \cup P_2$ gives a shortest path between u and x that contains both uv and xy. Hence, e_1 and e_2 are conflicting. □

Lemma 5.66. *Let $e_1 \in E_{\mathsf{bridge}} \cup E_{\mathsf{even}}$ and $e_2 \in E_{\mathsf{rem}}$. Then any VSRC of G colors e_1 and e_2 with different colors.*

Proof. Let $e_1 = xy$ and $e_2 = uv$, let C be the odd cycle containing e_2, and let $w = \mathsf{g}(x, C)$. By Observation 5.58, $w = \mathsf{g}(y, C)$. In other words, $x, y \in \mathsf{S}(w, C)$. Note that w is not a degree-2 vertex, because there are at least two vertices in $\mathsf{S}(w, C)$. Hence, $w \neq \mathsf{vopp}(uv)$ by the definition of E_{rem}. Hence, by Lemma 5.60, w.l.o.g., there is a shortest path P_1 from w to u that contains uv.

We now consider two cases, depending on whether $e_1 \in E_{\mathsf{bridge}}$ or $e_1 \in E_{\mathsf{even}}$. First, suppose that $e_1 \in E_{\mathsf{bridge}}$. Since xy is a bridge, we can assume w.l.o.g. that any shortest path from x to w contains xy. Let P_2 be such a shortest path. By Observation 5.57, if we append a shortest path from x to w with a shortest path from w to u, we get a shortest path from x to u. Thus, $P_1 \cup P_2$ is a shortest path from x to u containing xy and uv. Hence, e_1 and e_2 are conflicting.

Suppose that $e_1 \in E_{\mathsf{even}}$. Let C' be the even cycle containing e_1. Let $z = \mathsf{g}(v, C')$. From Lemma 5.59, we can assume w.l.o.g. that there is a shortest path from z to x that contains xy. Let this shortest path be P_3. Let P_4 be a shortest path between w and z. By Observation 5.57, $P_3 \cup P_4 \cup P_1$ is a shortest path between x and u that contains xy and uv. Hence, e_1 and e_2 are conflicting. □

Lemma 5.67. *Let C_1 and C_2 be two distinct odd cycles and let $e_1 \in E(C_1) \cap E_{\mathsf{rem}}$ and $e_2 \in E(C_2) \cap E_{\mathsf{rem}}$. Then any VSRC of G colors e_1 and e_2 with different colors.*

Proof. Let $e_1 = xy$ and $e_2 = uv$, and let $w = \mathsf{g}(x, C_2)$. By Observation 5.58, $w = \mathsf{g}(y, C_2)$. Let $z = \mathsf{g}(u, C_1)$. By Observation 5.58, $z = \mathsf{g}(v, C_1)$. That is, $x, y \in \mathsf{S}(w, C_2)$ and $u, v \in \mathsf{S}(z, C_1)$. Note that w and z are not degree-2 vertices, because there are at least two vertices in $\mathsf{S}(w, C_2)$ and $\mathsf{S}(z, C_1)$. Hence, $w \neq \mathsf{vopp}(uv)$ and $z \neq \mathsf{vopp}(xy)$ by the definition of E_{rem}. Hence, by Lemma 5.60, we can assume w.l.o.g. that there is a shortest path P_1 from u to w that contains uv and there is a shortest path P_2 from z to x that contains xy. Let P_3 be a shortest path from w to z. By Observation 5.57, $P_1 \cup P_2 \cup P_3$ is a shortest path from x to u containing xy and uv. Hence, e_1 and e_2 are conflicting. □

Finally, we prove the existence of some non-conflicting pairs of edges.

Lemma 5.68. *For any $e_1 \in E_{\mathsf{opp}}$ and $e_2 \in \mathsf{OS}\,(e_1)$, e_1 and e_2 are not conflicting.*

Proof. Let $e_1 = uv$, $e_2 = xy$, and let C be the odd cycle containing e_1. For the sake of contradiction, suppose that uv and xy are conflicting. Assume w.l.o.g. that there is a shortest path P from x to v which contains uv and xy. From Observation 5.57, P contains a subpath P' from $\mathsf{g}(x, C)$ to v. Clearly, P' contains uv. Also, $\mathsf{g}(x, C) = \mathsf{vopp}(uv)$, because $x \in \mathsf{OS}(uv)$. However, recall that $\mathsf{vopp}(uv)$ is equidistant from u and v. Hence, any shortest path from $\mathsf{vopp}(uv)$ to v does not contain uv, which contradicts the existence of P'. □

5.5.3 Algorithm

Based on the results of the previous two subsections, we now describe the algorithm for cactus graphs. First, we color the edges of E_{bridge} with unique colors. By Lemma 5.62, no VSRC can use less colors to color E_{bridge}.

Next, we color the edges in E_{even} using colors that are distinct from those we used before. This will not harm the optimality of the constructed coloring, because of Lemma 5.63. Moreover, we use different colors for different even cycles, which does not harm optimality by Lemma 5.65. We then introduce a set of $\frac{|C|}{2}$ new colors for each even cycle C. For an opposite pair, we use the same color, and we color each opposite pair with a different color. Thus we use $\frac{|C|}{2}$ colors for each even cycle C. By Observation 5.64, no VSRC can use less colors to color C.

Next, we will color the edges in E_{rem} using colors that are distinct from those we used before. This will not harm the optimality of the constructed coloring, because of Lemma 5.66. For each odd cycle, we use a different set of colors. This will not harm the optimality of the constructed coloring, because of Lemma 5.67.

For each odd cycle C: If C is a 3-cycle we construct an auxiliary graph H_C for $E_{\mathsf{rem}} \cap C$ as follows. Let $V(H_C) = E_{\mathsf{rem}} \cap C$ and let $E(H_C) = \{e_1e_2 : e_1, e_2 \in V(H_C);\ e_1$ and e_2 are not conflicting in $G\}$.

Lemma 5.69. $\Delta(H_C) \leq 2$. *Also, H_C has no 3-cycle except when G is just a 3-cycle.*

Proof. It is easy to observe that in any odd cycle C, for any $e \in E(C)$, there are only two other edges in C that are not conflicting with e. Hence, $\Delta(H_C) \leq 2$. Now, suppose there is a 3-cycle in G. Then there exist edges e_1, e_2, e_3 in $E_{\mathsf{rem}} \cap C$ that are non-conflicting with each other. This is only possible if C is just a 3-cycle. Moreover, all the 3 edges in C are in E_{rem}. But then the whole graph is just C, which is a 3-cycle. □

Let M_C be a maximum matching of H_C. We can compute M_C in linear time, since $\Delta(H_C) \leq 2$. For an $e_1e_2 \in M_C$, color e_1 and e_2 with the same, new color. Then color each $e \in E_{\mathsf{rem}} \cap C$ that is unmatched in M_C, each using a new color.

Lemma 5.70. *The procedure for coloring $E_{\mathsf{rem}} \cap C$ gives a coloring of the edges in $E_{\mathsf{rem}} \cap C$ such that no conflicting edges are colored the same. Moreover, no VSRC of G can use less colors to color $E_{\mathsf{rem}} \cap C$ than used by the above procedure.*

Proof. Suppose two conflicting edges $e_1, e_2 \in E_{\mathsf{rem}} \cap C$ were colored the same. Then the corresponding vertices e_1 and e_2 were matched to each other in M_C. Hence, e_1 and e_2 are adjacent in H_C, meaning that e_1 and e_2 did not conflict each other in G, which is a contradiction. Hence, we have proved that no conflicting edges were given the same color by the procedure.

Now, consider any VSRC μ of G which colored $E_{\mathsf{rem}} \cap C$ with fewer colors than by our procedure. Observe that for any edge e in an odd cycle, there are only two other edges (say e_a and e_b) that are not conflicting with e. Moreover, e_a and e_b are conflicting with each other as H_C does not have any 3-cycle by Lemma 5.69 (we can forget about the case when G is 3-cycle as this is a trivial instance). This means that μ can use each color for at most two edges of $E_{\mathsf{rem}} \cap C$. Suppose there are k_1 colors that are assigned to two edges in $E_{\mathsf{rem}} \cap C$ by μ. Each pair of edges colored the same should be non-conflicting and hence have an edge between them in H_C. So, taking all pairs colored the same induces a matching of size k_1 of H_C. Then $k_1 \leq |M_C|$, because M_C is a maximum matching of H_C. But then the number of colors used by μ is equal to $k_1 + (|E_{\mathsf{rem}} \cap C| - 2k_1) = |E_{\mathsf{rem}} \cap C| - k_1$. The number of colors used by our procedure is $|M_C| + |E_{\mathsf{rem}} \cap C| - 2|M_C| = |E_{\mathsf{rem}} \cap C| - |M_C| \leq |E_{\mathsf{rem}} \cap C| - k_1$. Hence, we use at most the number of colors used by μ. □

Finally, we color the edges of E_{opp} without introducing new colors. Indeed, for every $e \in E_{\mathsf{opp}}$, it follows from Lemma 5.61 that there exists an edge $e' \in E(\mathsf{OS}(e)) \cap (E_{\mathsf{bridge}} \cup E_{\mathsf{even}} \cup E_{\mathsf{rem}})$, which does not conflict with e by Lemma 5.68. Since e' is already colored, say by color c, then we can simply re-use that color c for e. Indeed, suppose for the sake of contradiction that there is a shortest path P between two vertices x, y that contains e and that contains another edge e'' using the color c. By Lemma 5.68, $e'' \notin \mathsf{OS}(e)$. This implies that $e'' \notin E_{\mathsf{bridge}} \cup E_{\mathsf{even}} \cup E_{\mathsf{rem}}$ by the choice of c and the construction of the coloring. Hence, $e'' \in E_{\mathsf{opp}}$. However, by a similar argument, e'' can receive color c only if $e' \in \mathsf{OS}(e'')$. But then, either $\mathsf{OS}(e) \subseteq \mathsf{OS}(e'')$ or $\mathsf{OS}(e'') \subseteq \mathsf{OS}(e)$, and thus e and e'' are not conflicting by Lemma 5.68, a contradiction to the existence of P.

Proof of Theorem 5.6: It follows from the above discussion that the constructed coloring is a very strong rainbow coloring of G. Moreover, it uses $\mathsf{vsrc}(G)$ many colors. Also, it is clear that the coloring can be computed in polynomial time. □

5.6 Other Algorithmic Results for VSRC

In this section, we first show that 2-VSRC can be solved in polynomial time. Then we show that k-VSRC is fixed parameter tractable when parameterized by $k + \mathsf{tw}(G)$, where $\mathsf{tw}(G)$ denotes the treewidth of G.

For proving both the results, we use an auxiliary graph G' defined as follows: add a vertex v_e to G' for each edge e in G; add an edge between vertices v_{e_1} and v_{e_2} in G' if and only if edges e_1 and e_2 appear together in some shortest path of G. The latter condition can be easily checked in polynomial time. Observe that $\mathsf{vsrc}(G) \leq k$ if and only if G' admits a proper k-coloring. Since 2-Coloring is solvable in polynomial time, this implies that 2-VSRC is polynomial time solvable and hence we have proved Proposition 5.5.

It is worth noting that the chromatic number of the auxiliary graph G' constructed in the above proof always corresponds to the very strong rainbow connection number of G. However, in the transformation from G to G', we lose a significant amount of structural information. For example, if G is a path or a star ($\mathsf{tw}(G) = 1$), then G' is a clique ($\mathsf{tw}(G') = |V(G'| - 1 = |V(G)| - 2$), where we use $\mathsf{tw}(G)$ to denote the treewidth of G. If $\mathsf{vsrc}(G) \leq k$, then we can prove that $|V(G')| \leq k^{(k+1)} \cdot (\mathsf{tw}(G)+1)^{(k+1)}$ as shown below.

Lemma 5.71. *Let G be any connected graph and let $\mathsf{vsrc}(G) \leq k$ and $\mathsf{tw}(G) \leq t-1$. Then $\Delta(G) \leq kt$ and $|V(G)| \leq (kt)^{k+1}$.*

Proof. By Lemma 5.53, the fact $\mathsf{vsrc}(G) \leq k$ implies that G is k-perfectly groupable. Hence, the neighborhood of each vertex can be partitioned into k or fewer cliques. Since $\mathsf{tw}(G) \leq t-1$, each clique of G has size at most t [141]. Hence, $\Delta(G) \leq kt$. Now observe that $\mathsf{vsrc}(G) \leq k$ implies that the diameter of G is at most k. Combined, these two facts imply that $|V(G)| \leq (kt)^{k+1}$. □

Using the above lemma, we now prove that k-VSRC is fixed parameter tractable when parameterized by $k + \mathsf{tw}$, thereby proving Theorem 5.7.

Proof of Theorem 5.7: Let $\mathsf{vsrc}(G) \leq k$ and $\mathsf{tw}(G) \leq t-1$. We now construct the auxiliary graph G' as above. Now, we only need to compute the chromatic number of G'. We aim to use the algorithm by Björklund et al. [22] which computes the chromatic number of a graph on n vertices in $2^n n^{\mathcal{O}(1)}$ time. To bound $|V(G')|$, we observe that by Lemma 5.71, $|V(G)| \leq (kt)^{k+1}$ and $\Delta(G) \leq kt$. Hence, $|V(G')| = |E(G)| \leq (kt)^{(k+2)}$. Therefore, the chromatic number of G', and thereby $\mathsf{vsrc}(G)$, can be determined in $\mathcal{O}(2^{(kt)^{(k+2)}}(kt)^{\mathcal{O}(k+2)})$ time. □

5.7 RVC and SRVC in Bipartite graphs and their subclasses

In this section, we show that k-RVC and k-SRVC are hard on bipartite graphs for $k \geq 3$. We complement these results by showing that both problems can be solved in linear time on bipartite permutation graphs. We first observe that computing $\mathsf{rvc}(G)$ or $\mathsf{srvc}(G)$ is easy on bipartite graphs of diameter 3. The same observation was made by Li et al. [108].

Proposition 5.72 ([108])**.** *If G is a bipartite graph with $\mathsf{diam}(G) = 3$, then $\mathsf{rvc}(G) = \mathsf{srvc}(G) = 2$. Moreover, such a coloring can be found in linear time.*

Proof. The statement follows from Proposition 5.12 and the fact that every bipartite graph has a proper 2-coloring that can be found in linear time. □

It turns out that if $\mathsf{diam}(G) \geq 4$, then $\mathsf{rvc}(G)$ and $\mathsf{srvc}(G)$ of a bipartite graph G become much harder to compute, as claimed in Theorem 5.9, which we state here again for convenience.

Theorem 5.9. *Let G be a bipartite graph of diameter 4. It is NP-complete to decide both whether $\mathsf{rvc}(G) \leq k$ and whether $\mathsf{srvc}(G) \leq k$, for every $k \geq 3$. Moreover, it is NP-hard to approximate both $\mathsf{rvc}(G)$ and $\mathsf{srvc}(G)$ within a factor of $n^{1/3-\varepsilon}$, for every $\varepsilon > 0$.*

We now prove Theorem 5.9. Towards this, we provide the following construction. For an illustration of the construction, see Figure 5.10.

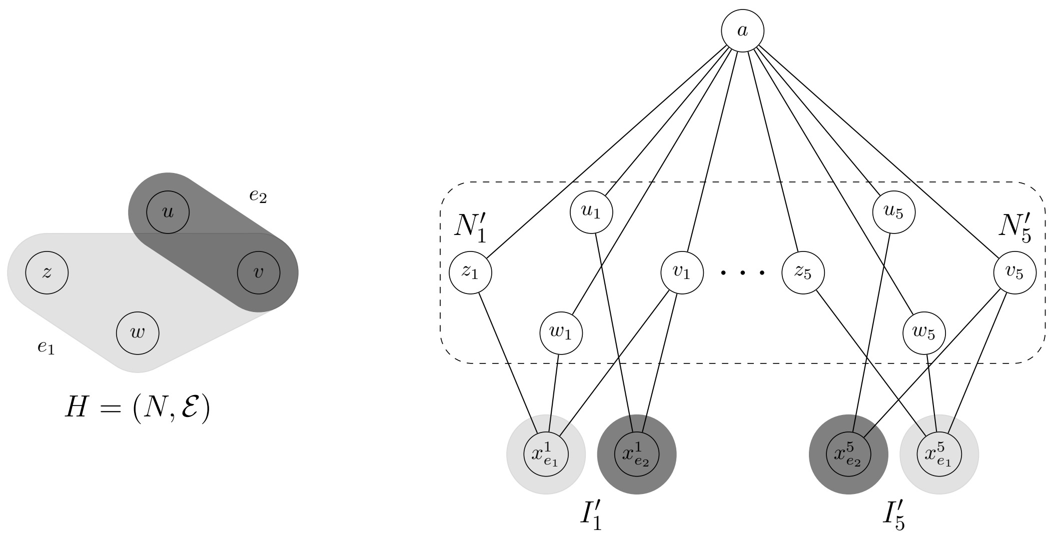


Figure 5.10: A hypergraph $H = (N, \mathcal{E})$ (left) transformed into a bipartite graph G (right) as described in the proof of Lemma 5.73. The dashed rectangle with rounded corners contains the sets in N'.

Lemma 5.73. *Let H be a hypergraph on n vertices. Then in polynomial time we can construct a bipartite graph G of diameter 4 and with $\mathcal{O}(n^3)$ vertices such that for any $k \in [n]$, H has a proper k-coloring if and only if G has a $(k+1)$-coloring under which G is (strongly) rainbow vertex-connected. Moreover, if H is a planar graph, then G is an apex graph.*

Proof. Let $H = (N, \mathcal{E})$ be an arbitrary hypergraph and let $n = |N|$. We construct a bipartite graph $G := (V, E)$ where $V := \{a\} \cup N' \cup I'$, $N' := N'_1 \cup \cdots \cup N'_{n+1}$, $N'_i := \{v_i \mid v \in N\}$, $I' := I'_1 \cup \cdots \cup I'_{n+1}$, $I'_i := \{x^i_e \mid e \in \mathcal{E}\}$ and $E := \{av \mid v \in N'\} \cup \{v_i x^i_e \mid e \in \mathcal{E}, i \in [n+1], v \in e\}$. A bipartition of G is given by $(\{a\} \cup I',\ N')$. Observe that $\mathsf{diam}(G) = 4$ and that G has $\mathcal{O}(n^3)$ vertices. Moreover, if H is a planar graph, then G consists of vertex a plus $n+1$ copies of the graph obtained from H by subdividing each edge of H, and thus G is an apex graph. For an illustration of the construction, see Figure 5.10.

Consider any proper k-coloring $h : N \to [k]$ of H, i.e., no hyperedge of H is monochromatic under h. We construct a coloring $c : V \to [k+1]$ in the following way. First, for every $v \in N$, we give the vertices $v_1, v_2, \ldots, v_n$ of G the same color as v, i.e., $c(v_i) = h(v)$ for all $v \in N$ and $i \in [n+1]$. We give vertex a the color $k+1$, i.e., $c(a) = k+1$. The vertices in I all receive the same color, which is any arbitrary color in $[k+1]$. Now we prove that G is strongly rainbow vertex connected under c by showing that there is a rainbow vertex shortest path between every pair of vertices. The only non-trivial case is when both vertices of the pair are in I. Consider two distinct vertices $x^i_e, x^j_f \in I$ (it is possible that $e = f$ or $i = j$ but not both). Since e and f are not monochromatic under h, we can pick two distinct vertices $u \in e$ and $v \in f$ such that $h(u) \neq h(v)$. It is clear that the path $x^i_e uav x^j_f$ is a shortest path between x^i_e and x^j_f and that it is a rainbow vertex path. Hence, G is strongly rainbow vertex-connected under c.

Conversely, let c be a $(k+1)$-coloring of G under which G is (strongly) rainbow vertex-connected. For each $i \in [n+1]$, define h_i to be the vertex coloring of H such that $h_i(v) = c(v_i)$ for all $v \in N$. Let M_i be the set of vertices $v \in N$ such that $h_i(v) \neq c(a)$. Let $h'_i(v) = h_i(v)$ if $v \in M_i$ and $h'_i(v) = 1$ otherwise. It is clear that each h'_i is a k-coloring of H. We claim that there exists an $i \in [n+1]$ such that h'_i is a proper k-coloring of H. For the sake of contradiction, suppose that h'_i is not a proper k-coloring of H for every $i \in [n+1]$. For each $i \in [n+1]$, let $e_i \in \mathcal{E}$ be a monochromatic edge under h'_i. Let c_i be the color of vertices in e_i under h'_i. Now, since $k \leq n$, there exist distinct $i, j \in [n+1]$ such that $c_i = c_j$. From the construction of h' and definition of e_i, all vertices in $V_1 := \{v_i : v \in e_i\}$ are colored either $c(a)$ or c_i under c. Similarly, all vertices in $V_2 := \{v_j : v \in e_j\}$ are colored either $c(a)$ or $c_j = c_i$ under c. Note that V_1 is the set of vertices adjacent to x_{e_i} and V_2 is the set of vertices adjacent to x_{e_j}. Also, $V_1 \cap V_2 = \emptyset$ as $i \neq j$. Consider any (shortest) path from x_{e_i} to x_{e_j}. The internal vertices of the path consists of 3 vertices, one from V_1, the vertex a, and one vertex from V_2. But these vertices are all colored with either $c(a)$ or $c_i = c_j$. Hence the path is not a rainbow vertex path and thus c is not a (strong) rainbow coloring. Hence, we have a contradiction. □

Proof of Theorem 5.9. For membership in NP, a certificate that $\mathsf{rvc}(G) \leq k$ (or $\mathsf{srvc}(G) \leq k$) consists of a k-coloring and a list of (shortest) paths, one for every pair of vertices, that are rainbow vertex connected. For NP-hardness, we observe that the transformation of Lemma 5.73 implies a straightforward reduction from Hypergraph k-Coloring. Since Hypergraph k-Coloring is NP-complete for each $k \geq 2$, this proves the first part of the theorem.

For the second part of the theorem, we consider an instance of Coloring that consists of a graph on ℓ vertices and apply Lemma 5.73. Note that the total number of vertices in G is $n = \mathcal{O}(\ell^3)$. From the hardness of approximation of Coloring, we know that for all $\varepsilon > 0$, it is NP-hard to distinguish between the case when H is properly colorable with ℓ^ε colors and the case when H is not properly colorable with fewer than $\ell^{1-\varepsilon}$ colors [164]. By Lemma 5.73, this implies that it is NP-hard to distinguish between the case when G is (strong) rainbow vertex colorable with $\ell^\varepsilon + 1 \leq n^\varepsilon + 1$ colors and the case when G is not (strong) rainbow vertex colorable with fewer than $\ell^{1-\varepsilon}+1 = \Omega(n^{1/3-\varepsilon})$ colors. The second statement of the theorem follows. □

Using the same construction, we now proceed to give a proof of Theorem 5.8, which we state here again for convenience.

Theorem 5.8. *Let G be a bipartite apex graph of diameter 4. It is NP-complete to decide both whether $\mathsf{rvc}(G) \leq 4$ and whether $\mathsf{srvc}(G) \leq 4$. Moreover, it is NP-hard to approximate $\mathsf{rvc}(G)$ and $\mathsf{srvc}(G)$ within a factor of $5/4 - \varepsilon$, for every $\varepsilon > 0$.*

Proof. The proof follows along the same lines as the proof of the first part of Theorem 5.9. Instead of Hypergraph k-Coloring, however, we reduce from Planar 3-Coloring, the problem of deciding whether a planar graph has a proper 3-coloring. This problem is NP-complete. The NP-completeness follows from Lemma 5.73 because, the graph resulting from the construction is a bipartite apex graph of diameter 4, when we start with a planar graph.

For the hardness of approximation, we recall that any planar graph has a proper 4-coloring, and thus the graph G constructed in Lemma 5.73 has a 5-coloring under which G is rainbow vertex-connected. Hence, Lemma 5.73 combined with the NP-hardness of Planar 3-Coloring makes it NP-hard to decide whether G has a 5-coloring or a 4-coloring under which G is rainbow vertex-connected. The required hardness result follows from this. $\square$

We now complement the above hardness results with a positive result in the case when a bipartite graph is also a permutation graph, as claimed in Theorem 5.11. It has been shown by Spinrad et al. [147] that a bipartite graph is a permutation graph if and only if it has a *strong ordering* (see preliminaries in Section 5.1.5 for definition), and such an ordering can be computed in linear time. The following property of the BFS tree of a bipartite permutation graph is well-known and easy to deduce from a strong ordering [156]. In every bipartite permutation graph G, it is possible to find a vertex v such that the levels $L_0, L_1, L_2, \ldots$ of the tree resulting from a BFS starting from v have the following properties. For all i, $L_0 = \{v\}$, L_i is an independent set and $G[L_i \cup L_{i+1}]$ is a *chain graph* (see preliminaries in Section 5.1.5 for definition). Moreover, for each level i, there exists a special vertex $a_i \in L_i$ such that $L_{i+1} \subset N(a_i)$. The vertex v can be picked as the first vertex of a strong ordering.

Lemma 5.74. *For $1 \leq i \leq j$, and for $x \in L_i$ and $y \in L_j$, $\mathsf{dist}(x, y)$ is either $j - i$ or $j - i + 2$.*

Proof. Clearly $\mathsf{dist}(x, y) \geq j - i$ by the property of BFS tree. There is a path of length $j - i + 2$ from x to y, namely the path $x a_{i-1} a_i a_{i+1} \ldots a_{j-1} a_j$. We will now prove that there is no path of length $j - i + 1$ from x to y, thereby completing the proof. Suppose there was such a path P. Observe that if z_2 is the proceeding vertex after z_1 in P, then z_2 and z_1 cannot be in the same level as each L_i is an independent set. Also, z_2 cannot be in a smaller level than z_1, as then the path P would have length at least $j - i + 2$. But then P has length $j - i$, a contradiction. $\square$

Theorem 5.75. *If G is a bipartite permutation graph, then $\mathsf{rvc}(G) = \mathsf{srvc}(G) = \mathsf{diam}(G) - 1$, and the corresponding (strong) rainbow vertex coloring can be found in time that is linear in the size of G.*

Proof. Let $G = (V, E)$ be a bipartite permutation graph. Let v be a first vertex in a strong ordering for G. We start by doing a BFS on G with v as the root. Let k be the number of levels in the BFS tree in addition to level 0. Hence, L_i is the set of vertices in level i of the BFS tree, $0 \leq i \leq k$, with $L_0 = \{v\}$. Since $\mathsf{dist}(v, y) = k$ for every $y \in L_k$, we conclude that $\mathsf{diam}(G) \geq k$. Also, for $1 \leq i \leq j$, and for any vertices $x \in L_i$ and $y \in L_j$, there is a path of length at most $j - i + 2 \leq k + 1$ due to Lemma 5.74. Hence, $\mathsf{diam}(G) \in \{k, k + 1\}$. We distinguish between these two cases:

Case 1. $\mathsf{diam}(G) = k$.

We construct a strong rainbow vertex coloring $c : V \to [k - 1]$ for G in the following way. If $x \in L_i$, we define $c(x) = i$, for $1 \leq i \leq k - 1$. We define $c(v) = k - 1$, and we give arbitrary colors between 1 and $k - 1$ to the vertices of L_k. To see that G is indeed rainbow vertex connected under c, consider any pair $x, y \in V$.

(1) $x = v$ and $y \in L_j$: Then the path $v a_1 \ldots a_{j-1} y$ is shortest and it is a rainbow vertex path.

(2) $x \in L_1$ and $y \in L_k$: In this case, $\mathsf{dist}(x, y) = k - 1$, because otherwise we would have $\mathsf{dist}(x, y) = k + 1$ due to Lemma 5.74, which contradicts our assumption that $\mathsf{diam}(G) = k$. Since $\mathsf{dist}(x, y) = k - 1$, every shortest path between x and y is a rainbow vertex path, as every vertex of such a shortest path has to be in a distinct level of the BFS tree.

(3) $x \in L_1$ and $y \in L_j$ with $1 \leq j \leq k - 1$: If $\mathsf{dist}(x, y) = j - 1$, then again by the same argument used above, every shortest path between x and y is a rainbow vertex path. Otherwise, $\mathsf{dist}(x, y) = j + 1$ by Lemma 5.74, and the shortest path $xva_1 \ldots a_{j-1}y$ has distinct colors on all its internal vertices. (Note that y might have the same color as v if $j = k - 1$, but this is fine since y is the end of the path.)

(4) $x \in L_i$ and $y \in L_j$ with $2 \leq i \leq j \leq k$: If $\mathsf{dist}(x, y) = j - i$, then every shortest path is a rainbow vertex path as there can be only one vertex from each level. If $\mathsf{dist}(x, y) = j - i + 2$, the path $xa_{i-1}a_i \ldots a_{j-1}y$ is a rainbow vertex path and has length $j - i + 2$, and it is therefore shortest.

Case 2. $\mathsf{diam}(G) = k + 1$.

We construct a strong rainbow vertex coloring $c : V \to [k]$ for G in the following way. If $x \in L_i$, we define $c(x) = i$, for $1 \leq i \leq k - 1$. We define $c(v) = k$, and we give arbitrary colors between 1 and k to the vertices of L_k. To see that G is indeed rainbow-connected under c, consider any pair $x, y \in V$.

Assume without loss of generality that $x \in L_i$ and $y \in L_j$, with $0 \leq i \leq j \leq k$. If $\mathsf{dist}(x, y) = j - i$ then every shortest path between x and y is rainbow. Otherwise, the path $xa_{i-1}a_i \ldots a_{j-1}y$ is a rainbow vertex path and has length $j - i + 2$, therefore being shortest by Lemma 5.74.

In both cases, c is a strong rainbow vertex coloring for G with $\mathsf{diam}(G) - 1$ colors. By Proposition 5.12 we can conclude that $\mathsf{rvc}(G) = \mathsf{srvc}(G) = \mathsf{diam}(G) - 1$. It is easy to see from the proof that the coloring can be constructed in linear time. □

5.8 RVC and SRVC in Chordal graphs and their subclasses

In this section, we investigate the complexity of k-RVC and k-SRVC on chordal graphs and some subclasses of chordal graphs. We start by proving that both problems are NP-complete when the input graph is a split graph, implying that they are also NP-complete on chordal graphs. On the positive side, we show that k-RVC is polynomial-time solvable on interval graphs, and both k-RVC and k-SRVC are polynomial-time solvable on block graphs and on unit interval graphs.

We start by observing that computing $\mathsf{rvc}(G)$ or $\mathsf{srvc}(G)$ is easy on graphs of diameter 2.

Proposition 5.76 ([100]). *If G is a graph with $\mathsf{diam}(G) = 2$, then $\mathsf{rvc}(G) = \mathsf{srvc}(G) = 1$. Moreover, such a coloring can be found in linear time.*

Proof. Color each vertex of G with the same color. Since each shortest path between two vertices contains at most one internal vertex, G is strongly rainbow vertex-connected under this coloring. □

If G is a split graph having $\mathsf{diam}(G) = 3$ (note that split graphs have diameter at most 3), then $\mathsf{rvc}(G)$ and $\mathsf{srvc}(G)$ become much harder to compute, as claimed in Theorem 5.10, which we state again here for convenience.

Theorem 5.10. *Let G be a split graph of diameter 3. It is NP-complete to decide both whether $\mathsf{rvc}(G) \leq k$ and whether $\mathsf{srvc}(G) \leq k$, for every $k \geq 2$. Moreover, it is NP-hard to approximate both $\mathsf{rvc}(G)$ and $\mathsf{srvc}(G)$ within a factor of $n^{1/3-\varepsilon}$, for every $\varepsilon > 0$.*

Towards proving Theorem 5.10, we prove the following construction, which closely mimics the construction of Lemma 5.73.

Lemma 5.77. *Let H be a hypergraph on n vertices. Then, in polynomial time, we can construct a split graph G of diameter 3 and with $\mathcal{O}(n^3)$ vertices such that for any $k \in [n]$, H has a proper k-coloring if and only if G has a k-coloring under which G is (strongly) rainbow vertex-connected.*

Proof. Let $H = (N, \mathcal{E})$ be an arbitrary hypergraph and let $n = |N|$. We construct a split graph $G = (N' \cup I',\ E)$ where $N' = N'_1 \cup \cdots \cup N'_{n+1}$, $I' = I'_1 \cup \cdots \cup I'_{n+1}$, $N'_i := \{v_i \mid v \in N\}$, $I'_i := \{x^i_e \mid e \in \mathcal{E}\}$ and $E := \{u_i v_j \mid u, v \in N,\ i, j \in [n+1]\} \cup \{v_i x^i_e \mid v \in N, e \in \mathcal{E}, i \in [n+1], v \in e\}$. Let $V = N' \cup I'$. The constructed graph G is a split graph since $G[I']$ is an independent set and $G[N']$ is a clique. Observe that $\mathsf{diam}(G) = 3$ (except when H has a vertex that is contained in all hyperedges, in which case the diameter is 2; in this case H has an easy proper 2-coloring; so we can assume that this case does not occur) and that G has $\mathcal{O}(n^3)$ vertices.

Consider any proper k-coloring $h : N \to [k]$ of H, i.e., no hyperedge of H is monochromatic under h. We construct a coloring $c : V \to [k]$ in the following way. First, for every $v \in N$, we give the vertices $v_1, v_2, \ldots, v_n$ of G the same color as v, i.e., $c(v_i) = h(v)$ for all $v \in N$ and $i \in [n+1]$. The vertices in I all receive the same color, which is any arbitrary color in $[k]$. Now, we prove that G is strongly rainbow vertex-connected under c by showing that there is a rainbow vertex shortest path between every pair of vertices. The only non-trivial case is when both vertices of the pair are in I. Consider two distinct vertices $x^i_e, x^j_f \in I$ (it is possible that $e = f$ or $i = j$ but not both). Since e and f are not monochromatic under h, we can pick two distinct vertices $u \in e$ and $v \in f$ such that $h(u) \neq h(v)$. It is clear that the path $x^i_e u v x^j_f$ is a shortest path between x^i_e and x^j_f and that it is rainbow vertex path.

Conversely, let c be a k-coloring of G under which G is (strongly) rainbow vertex-connected. For each $i \in [n+1]$, define h_i to be the vertex coloring of H such that $h_i(v) = c(v_i)$ for all $v \in N$. We claim that there exists an $i \in [n+1]$ such that h_i is a proper k-coloring of H. For the sake of contradiction, suppose that h_i is not a proper k-coloring of H for every $i \in [n+1]$. For each $i \in [n+1]$, let $e_i \in \mathcal{E}$ be a monochromatic edge under h_i. Let c_i be the color of vertices in e_i under h_i. Now, since $k \leq n$, there exist distinct $i, j \in [n+1]$ such that $c_i = c_j$. From the construction of h and definition of e_i, all vertices in $V_1 := \{v_i : v \in e_i\}$ are colored c_i under c. Similarly, all vertices in $V_2 := \{v_j : v \in e_j\}$ are colored $c_j = c_i$ under c. Note that V_1 is the set of vertices adjacent to x_{e_i} and V_2 is the set of vertices adjacent to x_{e_j}. Also, $V_1 \cap V_2 = \emptyset$ as $i \neq j$. Consider any (shortest) path from x_{e_i} to x_{e_j}. The vertex adjacent to x_{e_i} in the path is from V_1, and the vertex adjacent to x_{e_i} in the path is from V_2, and they are distinct as $V_1 \cap V_2 = \emptyset$. But then both vertices colored with $c_i = c_j$. Hence the path is not a rainbow vertex path and thus c is not a (strong) rainbow coloring. Hence, we have a contradiction. □

Proof of Theorem 5.10. The proof follows in exactly the same way as Theorem 5.9, except that we apply Lemma 5.77 instead of Lemma 5.73. □

We now move on to the positive results. As a consequence of the following theorems, we complete the proof of Theorem 5.11.

Theorem 5.78. *Let G be a block graph, and let ℓ be the number of cut vertices in G. Then $\mathsf{rvc}(G) = \mathsf{srvc}(G) = \ell$. The corresponding (strong) rainbow vertex coloring can be found in time that is linear in the size of G.*

Proof. Let $G = (V, E)$ be a block graph and $\{a_1, a_2, \ldots, a_\ell\}$ be the set of cut vertices of G. We construct a strong rainbow vertex coloring $c : V \to [\ell]$ for G by defining $c(a_i) = i$ for $i \in [\ell]$ and giving the other vertices arbitrary colors between 1 and ℓ. An important property of block graphs is that there is a unique shortest path between every pair of vertices. Moreover, each internal vertex of such a path is a cut vertex. Since all the cut vertices received distinct colors, these shortest paths are all rainbow. The proof follows by observing that $\mathsf{rvc}(G) \geq \mathsf{srvc}(G) \geq \ell$ as well. □

Theorem 5.79. *If G is an interval graph, then $\mathsf{rvc}(G) = \mathsf{diam}(G) - 1$, and the corresponding rainbow vertex coloring can be found in time that is linear in the size of G.*

Proof. Let $G = (V, E)$ be an interval graph and $\mathcal{I}$ be an interval model for G. The interval corresponding to vertex v is denoted by I_v. For each interval $I \in \mathcal{I}$, we let $r(I)$ be its right endpoint and $\ell(I)$ its right endpoint. Let $I_u \in \mathcal{I}$ be such that $r(I_u) \leq r(I)$ for all $I \in \mathcal{I}$. Let $I_v \in \mathcal{I}$ be such that $\ell(I_v) \geq \ell(I)$ for all $I \in \mathcal{I}$. Let $P = ux_1x_2 \ldots x_kv$ be a shortest path between u and v in G. Observe that P is a connected dominating set. Furthermore, since P is a shortest path, $k \leq \mathsf{diam}(G) - 1$. By the choice of u and v, we have that $N(u) \subseteq N(x_1)$ and $N(v) \subseteq N(x_k)$. This implies that the set $\{x_1, x_2, \ldots, x_k\}$ is also a connected dominating set. By Proposition 5.13, G has a rainbow vertex coloring $c : V \to [k]$ with $c(x_i) = i$, and we can give all the other vertices arbitrary colors. □

Recall that an interval graph is a *unit interval graph* if it has an interval model in which every interval has the same length (or no interval properly contains another interval). Unit interval graphs have the same BFS tree structure as that of bipartite permutation graphs, with the single difference that every level of the BFS tree is a clique instead of an independent set [83].

Theorem 5.80. *If G is a unit interval graph, then $\mathsf{rvc}(G) = \mathsf{srvc}(G) = \mathsf{diam}(G) - 1$, and the corresponding (strong) rainbow vertex coloring can be found in time that is linear in the size of G.*

Proof. Let $G = (V, E)$ be a unit interval graph. Let v be the vertex corresponding to a first interval in an ordering of the intervals in the unit interval model of G by their right endpoints. Do a BFS on G with v as the root. Let L_i be the set of vertices in level i of the BFS tree for all $0 \leq i \leq k$, with $L_0 = \{v\}$. Recall that for $0 \leq i \leq k-1$, there exists a special vertex $a_i \in L_i$ such that $L_{i+1} \subset N(a_i)$.

Consider a vertex $u \in L_k$. A shortest path between v and u has $k-1$ internal vertices, which implies that $\mathsf{diam}(G) \geq k$. To construct a strong rainbow coloring $c : V \to [k-1]$, we assign, for $1 \leq i \leq k-1$, $c(x) = i$ if $x \in L_i$ and we give arbitrary colors to the vertices of L_k and to v.

To see that G is strongly rainbow vertex-connected under c, consider $x, y \in V$. If both x and y are in the same level of the BFS tree, then they are adjacent. So let us consider the case when $x \in L_i$ and $y \in L_j$, with $0 \leq i < j \leq k$. If there is a shortest path between x and y each of whose vertices is in a distinct level of the BFS tree, then this path is rainbow. If this is not the case, we consider the path $xa_ia_{i+1}\dots a_{j-1}y$. In this case, this path is a shortest path between x and y, and its internal vertices have distinct colors, since only x and a_i belong to the same level of the BFS. This proves that c is indeed a strong rainbow coloring for G with $\mathsf{diam}(G) - 1$ colors. □

5.9 Open Problems

There are a plethora of open questions related to rainbow coloring. We describe a few ones that are related to our work.

(1) We showed many upper bounds on $\mathsf{rc}, \mathsf{src}$, and vsrc. One of the most interesting open questions in the area of rainbow coloring is to prove the conjecture that $\mathsf{src}(G) \leq n-1$ for any graph G. For the best of our knowledge, even a $o(n^2)$ upper bound is not known.

(2) Can the additive factor of 2 in our bound $\mathsf{rc} \leq f + 2$ be removed, i.e., is it true that $\mathsf{rc} \leq f$?

(3) We showed polynomial time algorithms for computing vsrc of cactus graphs. It is an interesting question, whether there are polynomial time algorithms for computing src and rc of cactus graphs. Another related question is whether there is a polynomial time algorithm for computing vsrc of outerplanar graphs, which are an immediate superclass of cactus graphs.

(4) We have seen that for $\mathsf{src}, \mathsf{rvc}, \mathsf{srvc}$, and vsrc, there are strong polynomially big inapproximability results. For rc, the best known inapproximability result is a $(2-\varepsilon)$-inapproximability [41]. Is there a polynomial time algorithm that can approximate rc within a constant factor, or at least a logarithmic factor? As far as we know, the best know approximation factor in polynomial time for an n-vertex graph is $\mathcal{O}(\sqrt{n})$ which follows from an $\mathcal{O}(r)$ approximation algorithm by Basavaraju et al. [18], where r is the radius of the graph.

CHAPTER 6

Spanning Tree Congestion and Connected Partitioning

6.1 Introduction

Graph sparsification/compression is the transformation of a *large* input graph into a *smaller/sparser* graph that preserves certain feature (e.g., distance, cut, congestion, flow) either exactly or approximately. The algorithmic significance of such a transformation is clearly evident, since the smaller graph might be used as a preprocessed input to an algorithm, so as to reduce subsequent running time and memory requirement. In this chapter, we focus on a natural problem in graph sparsification, called the spanning tree congestion (STC) problem. When we represent a graph by a spanning tree of it, each of the edges of the original graph is now represented by a path in the spanning tree. The number of such paths that pass through a tree-edge is called the *edge-congestion* of the tree-edge with respect to the spanning tree. The edge-congestion of the tree-edge is also equal to the number of edges of the original graph across the cut induced by the tree-edge in the spanning tree. Informally, the STC problem seeks a spanning tree of the input graph such that no tree edge has too much congestion.

The root of this problem dates back to at least 30 years ago [21, 144] with natural motivations from parallel computing and circuit design applications. The STC problem was first formally defined by Ostrovskii [132] in 2004, and since then a number of results about it have been published. The probabilistic version of the STC problem, coined as *probabilistic capacity mapping*, where one is allowed to have a distribution of spanning trees instead of one spanning tree, has also been studied in the literature, for example by Räcke [136].

Closely related to finding spanning tree with small congestion is the problem of partitioning the graph into balanced parts such that each of the parts are connected. The idea is that this allows one to recurse on the connected parts and look for spanning trees there. This motivated us to study the problem of *connected k-partition*, where we need to partition the graph into k parts, each of which are connected, and also each part has some specified size (or weight).

6.1.1 Connected Graph Partitioning and the Generalized Győri-Lovász Theorem.

Graph partitioning/clustering is a prominent topic in graph theory/algorithms, and has a wide range of applications. A popular goal is to partition the vertices into sets such that the number of edges across different sets is *small*. While the *min-sum* objective, i.e., minimizing the total number of edges across different sets, is more widely studied, in various applications, the more natural objective is the *min-max* objective, i.e., minimizing the maximum number of edges leaving each set. The min-max objective is our focus here. Depending on the application, there are often additional constraints on the sets in the partition. One such constraint is the *balancedness* constraint, which requires that the partitions have roughly equal sizes. Such a constraint often appears in *domain decomposition* in parallel computing. Another constraint on the partitions could be that of induced-connectivity. A partition satisfying this constraint is defined as follows.

Definition 6.1. *In a graph $G = (V, E)$, a* **connected k-partition** *is a partition of V into k subsets $V_1, V_2, \ldots, V_k$, such that for each $j \in [k]$, $G[V_j]$ is connected.*

One motivation for the induced-connectivity constraint is the divide-and-conquer approach for spanning tree construction. Such algorithms divide the graph into connected subgraphs and then recurse on them. Typically, balancedness constraint is also required by such algorithms, in order to obtain a balanced recursion tree.

Unfortunately, imposing both balancedness and induced-connectivity constraints simultaneously is not feasible for every graph; for instance, consider the star graph with more than 6 vertices and suppose we require a balanced connected 3-partition. Thus, it is natural to ask, for which graphs do partitions satisfying both constraints exist? One such sufficient condition was given by the following beautiful theorem proved independently by Győri and Lovász. The theorem in fact allows not just balanced partitions, but partitions into any specified sizes.

Theorem 6.2 (Győri-Lovász [118, 79]). *Let $G = (V, E)$ be a k-connected graph. Given any k distinct terminal vertices $t_1, \cdots, t_k$, and k positive integers $n_1, \cdots, n_k$ such that $\sum_{i=1}^{k} n_i = |V|$, there exists a connected k-partition of V into $V_1, V_2, \ldots, V_k$, such that for each $j \in [k]$, $t_j \in V_j$ and $|V_j| = n_j$.*

Lovász used advanced techniques from topology and homology theory which were non-constructive, whereas Győri used elementary graph theoretic techniques which were constructive. Recently, a generalization of the theorem to vertex weighted graphs were proved by Chen et al. [46].

Theorem 6.3 (Generalized Győri-Lovász theorem [46, Theorems 25, 26]). *Let $G = (V, E)$ be a k-connected graph. Let w be a weight function $w : V \to \mathbb{R}_{>0}$. For any $U \subset V$, let $w(U) := \sum_{v \in U} w(v)$. Given any k distinct vertices $t_1, \cdots, t_k$ of G, and k positive integers $T_1, \cdots, T_k$ such that for each $j \in [k]$, $T_j \geq w(t_j)$ and $\sum_{i=1}^{k} T_i = w(V)$, there exists a connected k-partition of V into $\cup_{j=1}^{k} V_j$, such that for each $j \in [k]$, $t_j \in V_j$ and $w(V_j) < T_j + \max_{v \in V} w(v)$.*

The proof by Chen et al. was an extension of the techniques from homology used by Lovász, and hence also non-constructive. Hence, the question remained whether the graph theoretic and constructive proof of Győri can be extended to the generalized Győri-Lovász theorem. We address this question in this work, and answer it positively.

We remark that the classical Győri-Lovász theorem follows from the generalized Győri-Lovász theorem by taking $w(v) = 1$ for all $v \in V$ and $T_j = n_j$ for all $j \in [k]$. We note that a perfect generalization where one requires that $w(V_j) = T_j$ is not possible. For example, there could be a vertex with weight more than all the T_j's. Hence an additive error of w_{max}, the largest weight, is inevitable in the theorem statement.

Theorem 6.3 can turn out to be a powerful tool for connected partitioning. By setting the weights to different parameters, it might be possible to obtain interesting results on connected partitioning. We briefly investigate few such immediate consequences of Theorem 6.3. We use one such consequence in proving one of our results for spanning tree congestion.

Our Contribution to Connected Partitioning

We give the first graph theoretic and constructive proof for the generalized Győri-Lovász theorem, by providing a *local search* algorithm.

Theorem 6.4. *(a) There is an algorithm that given a vertex-weighted k-connected graph, computes a connected k-partition satisfying the conditions stated in Theorem 6.3 (generalized Győri-Lovász Theorem), in $\mathcal{O}^*(2^n)$ time.*
(b) If we only need a connected $(\lfloor k/2 \rfloor + 1)$-partition instead of a connected k-partition, the running time of the algorithm improves to $\mathcal{O}^*(2^{\mathcal{O}((n/k)\log k)})$.

The algorithm and its analysis is given in Section 6.3. Our algorithm builds on the ideas in the graph theoretic approach used by Győri to prove the original Győri-Lovász theorem. An interesting observation is that the running time of our algorithm does not depend on the weights. We are not aware of any previous algorithm, even for the original unweighted version, that runs better than the brute force algorithm which runs over all possible k-partitions and takes $\Omega(2^{nk})$ time. Although Győri's proof for the unweighted version is implicitly algorithmic, no running time analysis of the algorithm has been done before, to the best of our knowledge.

Since Theorem 6.3 guarantees the existence of the required partition, the problem of computing such a partition is not a *decision problem* but a *search problem.* Our local search algorithm shows that this problem is in the complexity class PLS [88]; we raise its completeness in PLS as an open problem (see the open problems section at the end of the chapter).

We also prove few consequences of Theorem 6.4. First of all, for k-connected graphs, we obtain a connected k-partition such that the total-degree of each part is upper bounded by $4m/k$. This also implies that the min-max objective, i.e., the number of edges leaving any part, is also upper bounded by $4m/k$. The result is obtained by setting the weight of each vertex as its degree and applying Theorem 6.3.

Corollary 6.5. *Let G be a k-connected graph and $t_1, t_2, \ldots, t_k$ be distinct vertices in G.*
(a) We can find a connected k-partition of V into $\cup_{j=1}^{k} V_j$, such that for each $j \in [k]$, $t_j \in V_j$ and $\mathsf{deg}(V_j) < 4|E(G)|/k$ in $\mathcal{O}^*(2^n)$ time.
(b) We can find a connected $(\lceil k/2 \rceil + 1)$-partition of V into $\cup_{j=1}^{\lceil k/2 \rceil + 1} V_j$, such that for each $j \in [\lceil k/2 \rceil + 1]$, $t_j \in V_j$ and $\mathsf{deg}(V_j) < 8|E(G)|/k$ in $\mathcal{O}^*(2^{\mathcal{O}((n/k)\log k)})$ time.

Due to expander graphs, this bound is optimal up to a small constant factor. We remark that the above corollary is crucial for achieving one of our upper bound results for STC. Corollary 6.5 can be generalized to include approximate balancedness in terms of number of vertices by setting the weight of each vertex to be cm/n plus its degree in G. This gives the following corollary.

Corollary 6.6. *Given any fixed $c > 0$, if G is a k-connected graph with m edges and n vertices, then there exists a connected k-partition such that the total degree of vertices in each part is at most $(2c + 4)m/k$, and the number of vertices in each part is at most $\frac{2c+4}{c} \cdot \frac{n}{k}$.*

6.1.2 Spanning Tree Congestion

Given a connected graph $G = (V, E)$, let T be a spanning tree of it. For an edge $e = (u, v) \in E$, its *detour* with respect to T is the unique path from u to v in T; let $\mathsf{DT}(e, T)$ denote the set of edges in this detour. The **stretch** of e with respect to T is $|\mathsf{DT}(e, T)|$, the length of its detour. The **dilation** of T is $\max_{e \in E} |\mathsf{DT}(e, T)|$. The **edge-congestion** of an edge $e \in T$ is $\mathsf{ec}(e, T) := |\{f \in E : e \in \mathsf{DT}(f, T)\}|$, i.e., the number of edges in E whose detours contain e. The congestion of T is $\mathsf{cong}(T) := \max_{e \in T} \mathsf{ec}(e, T)$. The spanning tree congestion (STC) of the graph G is $\mathsf{STC}(G) := \min_T \mathsf{cong}(T)$, where T runs over all spanning trees of G.

We note that there is an equivalent cut-based definition for edge-congestion, which is usually more convenient to use. For each tree-edge $e \in T$, removing e from T results in two connected components; let U_e denote one of the components; then $\mathsf{ec}(e, T) := |E(U_e, V \setminus U_e)|$. Various types of congestion, stretch and dilation problems have been studied in computer science and discrete mathematics. In these problems, one typically seeks a spanning tree (or some other structure) with minimum congestion or dilation. Three of the well-known problems where minimization is done over all the spanning trees of the given graph are as follows:

(1) The *low-stretch spanning tree* (LSST) problem is to find a spanning tree which minimizes the total stretch of all the edges of G [6]. Note that minimizing the total stretch is equivalent to minimizing the sum of edge-congestions of the edges of the spanning tree.

(2) The *spanning tree congestion* (STC) problem is to find a spanning tree of minimum congestion [132].

(3) The *tree spanner problem* is to find a spanning tree of minimum dilation [29].

There are other congestion and dilation problems which do not seek a spanning tree, but some other structure. The most famous among them are the bandwidth and the cutwidth problems; see the survey [137] for more details.

Among the problems mentioned above, several strong results were published in connection with the LSST problem. Alon et al. [6] had shown an $\Omega(\max\{n \log n, m\})$ lower bound. Upper bounds have been derived and many efficient algorithms have been devised; the current best upper bound is $\tilde{\mathcal{O}}(m \log n)$ [6, 58, 1, 96, 2]. Since total stretch is identical to total edge-congestion, the best upper bound for the LSST problem automatically implies an $\tilde{\mathcal{O}}(\frac{m}{n} \log n)$ upper bound on the *average* edge-congestion. But in the STC problem, we concern the *maximum* edge-congestion; as we shall see, for some graphs, the maximum edge-congestion has to be a factor of $\tilde{\Omega}(\sqrt{n^3/m})$ larger than the average edge-congestion.

In comparison, there were not many strong and general results for the STC Problem, though it was studied extensively in the past 13 years. The problem was formally proposed by Ostrovskii [132] in 2004. Prior to this, Simonson [144] had studied the same parameter under a different name to approximate the cut width of outer-planar graph. A number of graph-theoretic results were presented on this topic [133, 105, 99, 98, 24]. Some complexity results were also presented recently [130, 23], but most of these results concern special classes of graphs. The most general result regarding STC of general graphs is an $\mathcal{O}(n\sqrt{n})$ upper bound by Löwenstein, Rautenbach and Regen in 2009 [119], and a matching lower bound by Ostrovskii in 2004 [132]. Note that the above upper bound is not interesting when the graph is sparse, since there is also a trivial upper bound of m. We come up with a strong improvement to these bounds, whose details will be given below.

The STC of many graph classes have been investigated as mentioned before. Estimating the STC of a random graph $\mathcal{G}(n,p)$ has also been looked at before. Note that STC is relevant for $\mathcal{G}(n,p)$ only when $p = \Omega(\frac{\log n}{n})$, because for smaller p, the graph is disconnected with very high probability. Ostrovskii [134] asked whether the STC of a random graph is $\Theta(n)$? Pogrow [135] obtained partial progress towards answering this question by showing that the STC of a random graph is at most $\mathcal{O}(n \operatorname{polylog} n)$ when $c_1 \log^3 n \le p \le c_2/\log n$ for some constants c_1 and c_2. We study the STC of random graphs further and obtain improved results as mentioned below. In particular, we completely settle the question asked by Ostrovskii.

Our Contribution to Spanning Tree Congestion

First, we prove a constructive upper bound on the STC of k-connected graphs by using our Corollary 6.5 from connected partitioning.

Theorem 6.7. *The spanning tree congestion of a k-connected graph G is at most $4|E(G)|/k$. Moreover, a spanning tree of G having congestion at most $8|E(G)|/k$ can be found in $\mathcal{O}^*(2^{(|V(G)|/k)\log k})$ time.*

We use the above theorem to obtain a constructive upper bound on the STC of general graphs as follows (see Section 6.4 for the algorithm and the proof).

Theorem 6.8. *For any connected graph $G = (V, E)$, there is an algorithm which computes a spanning tree with congestion at most $8\sqrt{mn}$ in $\mathcal{O}^*\left(2^{\mathcal{O}\left(n\sqrt{n/m}\log\sqrt{m/n}\right)}\right)$ time.*

In terms of average degree $d_{avg} = 2m/n$, we can state our upper bound as $\mathcal{O}(n\sqrt{d_{avg}})$. Observe that the upper bound is smaller than the trivial bound of m by a multiplicative factor of $\Theta\left(1/\sqrt{d_{avg}}\right)$, and smaller than the previous best known upper bound of $n\sqrt{n}$ by a multiplicative factor of $\Theta\left(\sqrt{d_{avg}/n}\right)$. Observe that the running time in terms of d_{avg} is $\mathcal{O}^*\left(2^{\mathcal{O}\left((n/\sqrt{d_{avg}})\log\sqrt{d_{avg}}\right)}\right)$. The running time is sub-exponential when d_{avg} is $\omega(1)$.

We also show that a spanning tree with only logarithmically worse congestion than the above upper bound can be constructed in polynomial time (see Section 6.4 for the algorithm and the proof).

Theorem 6.9. *There is an algorithm that given a connected graph $G = (V, E)$, computes a spanning tree of G having congestion at most $16\sqrt{mn}\log_2 n$ in time polynomial in the size of G.*

We also give lower bounds that asymptotically match the above upper bound for STC. For almost all ranges of average degree $2m/n$, we demonstrate graphs having $\Omega(\sqrt{mn})$ STC as follows (see Section 6.5 for the proof).

Theorem 6.10. *For any sufficiently large n, and for any m satisfying $n^2/2 \geq m \geq \max\{16n\log n, 100n\}$, there exists a connected graph with $N = (3 - o(1))n$ vertices and $M \in [m, 7m]$ edges, and having spanning tree congestion at least $\Omega(\sqrt{mn})$.*

Next, we show that for graphs with certain good expanding properties, there is a polynomial-time algorithm that computes a spanning tree with $\mathcal{O}(n)$ congestion. For this, first we introduce a class of graphs called (n, s, d_1, d_2, d_3, t)-expanding graphs.

Definition 6.11. *A graph $G = (V, E)$ on n vertices is an (n, s, d_1, d_2, d_3, t)-expanding graph if the following four conditions are satisfied:*

(1) for each $S \subseteq V(G)$ such that $|S| = s$, $|N(S)| \geq d_1 n$;

(2) for each $S \subseteq V(G)$ such that $|S| \leq s$, $|N(S)| \geq d_2|S|$;

(3) for each $S \subseteq V(G)$ such that $|S| \leq n/2$ and for any $S' \subset S$, $|N_{V\setminus S}(S')| \geq |S'| - t$.

(4) For each $S \subseteq V(G)$, $|E(S, V \setminus S)| \leq d_3|S|$.

Then, we develop a polynomial time algorithm for constructing a spanning tree of an (n, s, d_1, d_2, d_3, t)-expander with small congestion (see Section 6.6 for the algorithm and its analysis).

Theorem 6.12. *There is an algorithm that given an (n, s, d_1, d_2, d_3, t)-expanding graph, computes a spanning tree of it with congestion at most*

$$d_3 \cdot \left[4 \cdot \max\left\{ s + 1 \ , \ \left\lceil \frac{3d_1 n}{d_2} \right\rceil \right\} \cdot \left(\frac{1}{2d_1} \right)^{\log_{(2-\delta)} 2} + t \right], \quad \textit{where } \delta = \frac{t}{d_1 n}$$

and runs in time polynomial in n.

Then, we show that, with high probability, a random graph $\mathcal{G}(n, p)$ is an (n, s, d_1, d_2, d_3, t)-expanding graph with $s = \Theta(1/p)$, $d_1 = \Theta(1)$, $d_2 = \Theta(np)$, $d_3 = \Theta(np)$, $t = \Theta(1/p)$ (and hence $\delta = o(1)$). This implies the following upper bound for STC of random graphs (see Section 6.7 for the proof).

Theorem 6.13. *There is an algorithm that given a random graph $G = \mathcal{G}(n, p)$ for any $p \geq 64\log_2 n/n$, computes a spanning tree of G that has $\mathcal{O}(n)$ congestion with probability at least $1 - \mathcal{O}(1/n^2)$, and runs in time polynomial in n.*

We also give a matching lower bound for the STC of random graphs as follows (see Section 6.7 for the proof).

Theorem 6.14. *If $G \in \mathcal{G}(n, p)$ where $p \geq 32\log_2 n/n$, then the spanning tree congestion of G is $\Omega(n)$ with probability $1 - \mathcal{O}(1/n)$.*

The above two theorems together imply the following corollary.

Corollary 6.15. *For a random graph $\mathcal{G}(n, p)$ for any $p \geq 32\log_2 n/n$, the spanning tree congestion is $\Theta(n)$ with probability at least $1 - \mathcal{O}(1/n)$.*

The above result completely resolves an open problem raised by Ostrovskii [134] about the STC of random graphs.

6.1.3 Further Related Work.

Graph partitioning/clustering is a prominent research topic with wide applications, so it comes as no surprise that a lot of work has been done on various aspects of the topic; we refer readers to the two extensive surveys by Schaeffer [142] and by Teng [153]. Kiwi, Spielman and Teng [94] formulated the min-max k-partitioning problem and gave bounds for classes of graphs with *small separators*, which were then improved by Steurer [148]. On the algorithmic side, many of the related problems are NP-hard, so the focus is on devising approximation algorithms. Since the seminal work of Arora, Rao and Vazirani [13] on sparsest cut and of Spielman and Teng [146] on local clustering, graph partitioning/clustering algorithms with various constraints have attracted attention across theory and practice; we refer readers to [15] for a fairly recent account of the development. The min-sum objective has been extensively studied; the min-max objective, while striking as the more natural objective in some applications, has received much less attention. The only algorithmic work on this objective (and its variants) are Svitkina and Tardos [152] and Bansal et al. [15]. None of the above work addresses the induced-connectivity constraint.

The classical version of Győri-Lovász Theorem (i.e., the vertex weights are uniform) was proved independently by Győri [79] and Lovász [118]. Lovász's proof uses homology theory and is non-constructive. Győri's proof is elementary and is constructive implicitly, but he did not analyze the running time. Polynomial time algorithms for constructing connected k-partition were devised for $k = 2, 3$ [151, 158], but no non-trivial finite-time algorithm was known for general graphs with $k \geq 4$.[1] Recently, Hoyer and Thomas [85] provided a clean presentation of Győri's proof by introducing their own terminology, which we use for our constructive proof of Theorem 6.3.

Let BCP_k denote the maximum balanced connected k-partition problem and let WBCP_k denote the vertex weighted version. Here the objective function is the balancedness, which is defined as the size/weight of the smallest partition. Chlebíková [86] gave a polynomial time 4/3-approximation algorithm for WBCP_2, and also showed that there is no $n^{1-\varepsilon}$ additive approximation for the problem in polynomial time. Dyer and Frieze [54] showed BCP_k is NP-hard for all $k \geq 2$. Chataigner et al. [45] showed that there is no FPTAS for WBCP_k for any $k \geq 2$ even on k-connected graphs unless $\mathsf{P} = \mathsf{NP}$. They also gave 2-approximation algorithms for WBCP_3 and WBCP_4 on 3-connected and 4-connected graphs respectively, and showed that when k is part of the input, BCP_k does not admit a $(6/5-\varepsilon)$-approximation in polynomial time, unless $\mathsf{P} = \mathsf{NP}$. Recently Soltan, Yannakakis, and Zussman [145] studied the problem of doubly balanced connected partitioning, where you need balance within a partition with respect to a supply/demand function on vertices, and also balance among the partitions with respect to the size.

In 1987, Simonson [144] showed that for any outerplanar graph G, one can find in linear time a spanning tree that has congestion at most $\Delta(G) + 1$. Yosef Pogrow in his Master's thesis [135] showed that for edge weighted graphs, the spanning tree congestion (congestion of an edge is now defined as the ratio of the weight of edges across the cut and the weight of tree edge) is $\mathcal{O}(m)$ and also showed a matching lower bound. He also

[1] In 1994, there was a paper by Ma and Ma in *Journal of Computer Science and Technology*, which claimed a poly-time algorithm for all k. However, according to a recent study [84], Ma and Ma's algorithm can fall into an endless loop. Also, Győri said that the algorithm should be wrong (see [127]).

studied the spanning tree congestion of random graphs and showed that a random graph $\mathcal{G}(n,p)$ has STC at most $\tilde{\mathcal{O}}(n)$ with high probability for $\frac{c_1 \log^3 n}{n} \leq p \leq \frac{c_2}{\log n}$ where c_1 and c_2 are constants. If instead of a spanning tree, we need only a tree on the original vertices (the edges of the tree might not be in the graph), then the well-known Gomory-Hu trees minimizes the maximum congestion and they can be found in polynomial time [3].

Okamoto et al. [130] gave an $\mathcal{O}^*(2^n)$ algorithm for computing the exact STC of a graph. The probabilistic version of the STC problem, coined as *probabilistic capacity mapping*, is an important tool for several graph algorithm problems, e.g., the Min-Bisection problem. Räcke [136] showed that in the probabilistic setting, distance and capacity are *interchangeable*, which means that an upper bound for one objective implies the same upper bound for the other. Thus, due to the above-mentioned results on LSST, there is an upper bound of $\tilde{\mathcal{O}}(\log n)$ on the *maximum average congestion.* Räcke's result also implies an $\mathcal{O}(\log n)$ approximation algorithm to the Min-Bisection problem, improving upon the $\mathcal{O}(\log^{3/2} n)$ approximation algorithm of Feige and Krauthgamer [63]. However, in the deterministic setting, such interchanging phenomenon does not hold: there is a simple tight bound $\Theta(n)$ for dilation, but for congestion it can be as high as $\Theta(n\sqrt{n})$. For more background and key results on this topic, we recommend the paper by Andersen and Feige [9].

6.1.4 Technical Overview

To prove the generalized Győri-Lovász theorem constructively, we follow the same framework of Győri's proof [79], and we borrow terminology from the recent presentation by Hoyer and Thomas [85]. But it should be emphasized that proving our generalized theorem is not straight-forward, since in Győri's proof, at each stage a single vertex is moved from one set to other to make progress, while making sure that the former set remains connected. In our setting, in addition to this, we also have to ensure that the weights in the partitions do not exceed the specified limit; and hence any vertex that can be moved from one set to another need *not* be candidate for being transferred.

As mentioned before, a crucial ingredient for our upper bound results on STC is Theorem 6.7, which is a direct corollary of our constructive generalized Győri-Lovász theorem (Theorem 6.4). Theorem 6.7 takes care of the highly-connected cases; for other cases we provide a recursive way to construct a low congestion spanning tree; see Section 6.4 for details. For showing our lower bound for general graphs, the challenge is to maintain high congestion while keeping density small. To achieve this, we combine three expander graphs with *little* overlapping between them, and we further make those overlapped vertices of very high degree. This will force a tree-edge adjacent to the centroid of any spanning tree to have high congestion; see Section 6.5 for details.

We formulate a set of expanding properties and show that we can construct a spanning tree of better congestion guarantee in polynomial time for graphs satisfying those expanding properties. The basic idea is simple: start with a vertex v of high degree as the root. Now try to grow the tree by keep attaching new vertices to it, while keeping the invariant that the subtrees rooted at each of the neighbors of v are roughly balanced in size; each such subtree is called a *branch.* But when trying to grow the tree in a balanced way, we will soon realize that as the tree grow, all the remaining vertices may be seen to be adjacent only to a few number of "heavy" branches. To help the balanced growth, the algorithm will identify a *transferable* vertex which is in a heavy branch, and it and its descendants in the tree can be transferred to a "lighter" branch. Another technique is to use multiple rounds of matching between vertices in the tree and the remaining vertices to attach new vertices to the tree. This will tend to make sure that all subtrees do not grow uncontrolled. By showing that random graph satisfies the expanding properties with appropriate parameters, we show that a random graph has STC of $\Theta(n)$ with high probability.

6.2 Preliminaries

For any subset $U \subseteq V$, and a weight function w on the elements of V, we define $w(U) := \sum_{u \in U} w(u)$, and $w_{max} := \max_{v \in V} w(v)$. We use d_{avg} to denote the average degree of a graph.

6.3 Generalized Győri-Lovász Theorem

We prove Theorem 6.4 and its corollaries in this section. Let $G = (V, E)$ be a k-connected graph on n vertices and m edges, and $w : V \rightarrow \mathbb{R}_{>0}$ be a weight function.

6.3.1 Key Combinatorial Notions

We first highlight the key combinatorial notions used for proving Theorem 6.4; see Figures 6.1 and 6.2 for illustrations of some of these notions.

Fitted Partial Partition. First, we introduce the notion of *fitted partial partition* (FPP). An FPP A is a tuple of k subsets of V, $(A_1, \ldots, A_k)$, such that the k subsets are pairwise disjoint, and for each $j \in [k]$:

(1) $t_j \in A_j$,

(2) $G[A_j]$ is connected and

(3) $w(A_j) \leq T_j + w_{max} - 1$ (we say the set is *fitted* for satisfying this inequality).

We say that an FPP is a *Strict Fitted Partial Partition* (SFPP) if $A_1 \cup \cdots \cup A_k$ is a proper subset of V. We say the set A_j is *light* if $w(A_j) < T_j$, and we say that it is *heavy* otherwise. Note that there exists at least one light set in any SFPP, for otherwise $w(A_1 \cup \cdots \cup A_k) \geq \sum_{j=1}^{k} T_j = w(V)$, which means $A_1 \cup \cdots \cup A_k = V$. Also note that by taking $A_j = \{t_j\}$, we have an FPP, and hence at least one FPP exists.

Configuration. For a set A_j in an FPP A and a vertex $v \in A_j \setminus \{t_j\}$, we define the *reservoir* of v with respect to A, denoted by $R_A(v)$, as the vertices in the same connected component as t_j in $G[A_j \setminus v]$. Note that $v \notin R_A(v)$.

For a *heavy* set A_j, a sequence of vertices $(z_1, \ldots, z_p)$ for some $p \geq 0$ is called a *cascade* of A_j if $z_1 \in A_j \setminus \{t_j\}$ and $z_{i+1} \in A_j \setminus R_A(z_i)$ for all $1 \leq i < p$. The cascade is called a *null cascade* if $p = 0$, i.e., if the cascade is empty.

A *configuration* $\mathcal{C}_A$ is defined as a pair (A, D), where $A = (A_1, \cdots, A_k)$ is an FPP, and D is a set of cascades, which consists of exactly one cascade (possibly, a null cascade)

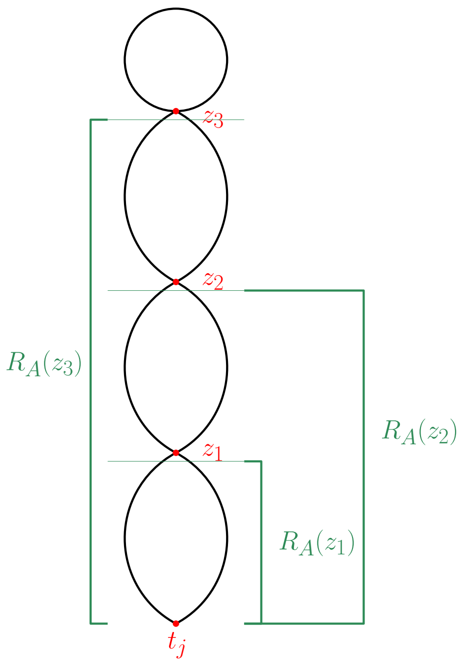


Figure 6.1: The figure shows a cascade (z_1, z_2, z_3) for the heavy set A_j and several reservoirs of the cascade vertices.
For any z_ℓ, note that $z_\ell \notin R_A(z_\ell)$. A cascade vertex z_ℓ (except possibly the last cascade vertex) is a cut-vertex of $G[A_j]$, i.e., $G[A_j \setminus \{z_\ell\}]$ is disconnected. The connected component of $G[A_j \setminus \{z_\ell\}]$ containing t_j is the reservoir of z_ℓ.
We identify $t_j = z_0$, but we clarify that a terminal vertex is never in a cascade. Each epoch between z_ℓ and $z_{\ell+1}$, and also the epoch above z_3, is a subset of vertices $B \subset A_j$, where $B \ni z_\ell$ and $G[B]$ is connected. Note that in general, it is possible that there is no vertex above the last cascade vertex.

for each heavy set in A. A vertex that is in some cascade of the configuration is called a *cascade vertex*.

Given a configuration, we define *rank* and *level* inductively as follows. (See Figure 6.2). Any vertex in a light set is said to have level 0. For $i \geq 0$, a cascade vertex is said to have rank $i+1$ if it has an edge to a level-i vertex but does not have an edge to any level-i' vertex for $i' < i$. A vertex u is said to have level i, for $i \geq 1$, if $u \in R_A(v)$ for some rank-i cascade vertex v, but $u \notin R_A(w)$ for any cascade vertex w such that rank of w is less than i. A vertex that is not in $R_A(v)$ for any cascade vertex v is said to have level ∞.

A configuration is called a *valid configuration* if for each heavy set A_j, rank is defined for each of its cascade vertices and the rank is strictly increasing in the cascade, i.e., if $\{z_1, \ldots, z_p\}$ is the cascade, then $\mathsf{rank}(z_1) < \cdots < \mathsf{rank}(z_p)$. Note that by taking $A_j = \{t_j\}$ and taking the null cascade for each heavy set (in this case A_j is heavy if $w(t_j) = T_j$), we get a valid configuration.

A vertex v of level ℓ is said to satisfy *maximality property* if each vertex adjacent on it is either a rank-$(\ell+1)$ cascade vertex, or has a level of at most $\ell+1$, or is one of the terminals t_j for some j. For any $\ell \geq 0$, a valid configuration is called an *ℓ-maximal*

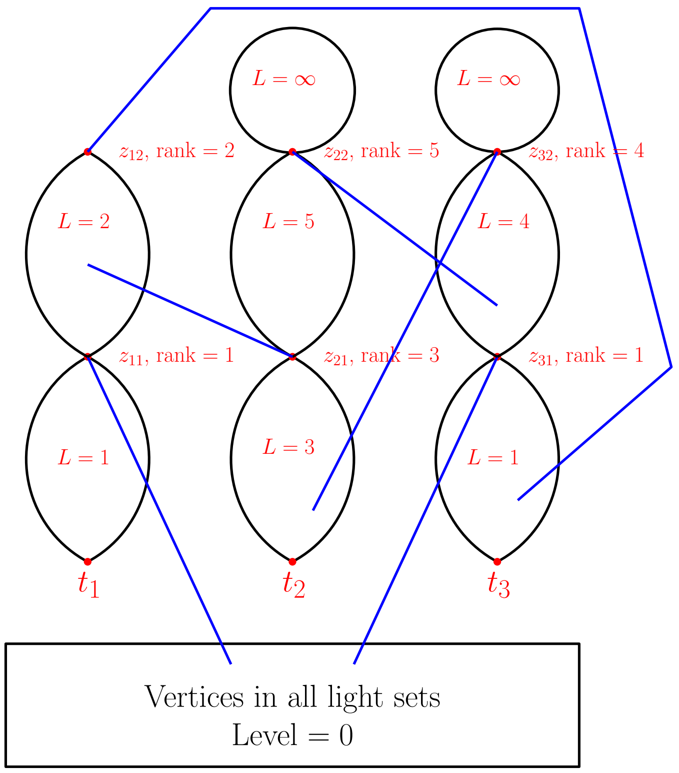


Figure 6.2: An instance of a valid configuration. Every blue segment/curve represent an edge from a cascade vertex to a vertex in some reservoir or light set.
Every cascade vertex connected to a light set has rank 1, and all vertices in the epoch immediately below a rank 1 cascade vertex are of level 1. Inductively, every cascade vertex connected to a vertex of level i has rank $i + 1$, and all vertices in the epoch immediately below a rank i cascade vertex are of level i. All vertices above the last cascade vertex of each cascade has level ∞.

configuration if all vertices having level at most $\ell - 1$ satisfy the maximality property. Note that by definition, any valid configuration is a 0-maximal configuration.

For a configuration $\mathcal{C}_A = ((A_1, \ldots, A_k), D)$, we define $S_A := V \setminus (A_1 \cup \cdots \cup A_k)$. An edge uv is said to be a *bridging-edge*[2] in $\mathcal{C}_A$ if $u \in S_A$, $v \in A_j$ for some $j \in [k]$, and $\text{level}(v) \neq \infty$.

A valid configuration $\mathcal{C}_A$ is said to be *ℓ-good* if the following conditions are all satisfied:

(1) highest rank of a cascade vertex in $\mathcal{C}_A$ is exactly ℓ (if there are no cascade vertices, then we take the highest rank as 0); note that this implies that the highest level other than ∞ is exactly ℓ,

(2) $\mathcal{C}_A$ is ℓ-maximal, and

(3) all bridging-edges uv in $\mathcal{C}_A$ (if any) are such that $u \in S_A$ and $\text{level}(v) = \ell$.

[2]We avoid the term bridge used by Hoyer and Thomas [85] to avoid confusion with the usual definition of bridges in graph theory

Note that taking $A_j = \{t_j\}$ and taking the null cascade for each heavy set gives a 0-good configuration.

Configuration Vectors and Their Total Ordering. For each configuration $\mathcal{C}_A = (A, D)$, we define a *configuration vector* as below:

$$(\ L_A \ , \ N_A^0 \ , \ N_A^1 \ , \ N_A^2 \ , \ \dots \ , \ N_A^n \ ),$$

where L_A is the number of light sets in A, and N_A^ℓ is the total number of all level-ℓ vertices in $\mathcal{C}_A$.

Next, we define an ordering on configuration vectors. Let $\mathcal{C}_A$ and $\mathcal{C}_B$ be configurations. We say $\mathcal{C}_A >_0 \mathcal{C}_B$ if

- $L_A < L_B$, or
- $L_A = L_B$, and $N_A^0 > N_B^0$.

We say $\mathcal{C}_A =_0 \mathcal{C}_B$ if $L_A = L_B$ and $N_A^0 = N_B^0$. We say $\mathcal{C}_A \geq_0 \mathcal{C}_B$ if $\mathcal{C}_A =_0 \mathcal{C}_B$ or $\mathcal{C}_A >_0 \mathcal{C}_B$. We say $\mathcal{C}_A =_\ell \mathcal{C}_B$ if $L_A = L_B$, and $N_A^{\ell'} = N_B^{\ell'}$ for all $\ell' \leq \ell$.

For $1 \leq \ell \leq n$, we say $\mathcal{C}_A >_\ell \mathcal{C}_B$ if

- $\mathcal{C}_A >_{\ell-1} \mathcal{C}_B$, or
- $\mathcal{C}_A =_{\ell-1} \mathcal{C}_B$, and $N_A^\ell > N_B^\ell$.

We say $\mathcal{C}_A \geq_\ell \mathcal{C}_B$ if $\mathcal{C}_A =_\ell \mathcal{C}_B$ or $\mathcal{C}_A >_\ell \mathcal{C}_B$. We say $\mathcal{C}_A > \mathcal{C}_B$ ($\mathcal{C}_A$ is *strictly better* than $\mathcal{C}_B$) if $\mathcal{C}_A >_n \mathcal{C}_B$.

6.3.2 Proof of Theorem 6.4

We first give two technical lemmas about ℓ-good configurations that we then use to prove Theorem 6.4.

Lemma 6.16. *Given any ℓ-good configuration*

$$\mathcal{C}_A = (A = (A_1, \dots, A_k), D_A))$$

that does not have a bridging-edge, we can find an $(\ell+1)$-good configuration

$$\mathcal{C}_B = (B = (B_1, B_2, \dots, B_k)\,, D_B)$$

in polynomial time such that $\mathcal{C}_B > \mathcal{C}_A$.

Proof. Since $\mathcal{C}_A$ is ℓ-good, $\mathcal{C}_A$ is also ℓ-maximal. Since $\mathcal{C}_A$ is ℓ-maximal, any vertex that is at level $\ell' < \ell$ satisfies the maximality property. So, for satisfying $(\ell+1)$-maximality, we only need to worry about the vertices that are at level ℓ. Let X_j be the set of all vertices $x \in A_j$ such that the following conditions are all satisfied:

(1) x is adjacent to a level-ℓ vertex,

(2) $\text{level}(x) \geq \ell + 1$ (i.e., $\text{level}(x) = \infty$ as the highest rank of any cascade vertex in an ℓ-good configuration is ℓ),

(3) $x \neq t_j$, and

(4) x is not a cascade vertex of rank ℓ.

For any j such that $X_j \neq \emptyset$, there is at least one vertex x_j such that $X_j \setminus \{x_j\} \subseteq R_A(x_j)$, from the definition of reservoir. And this vertex x_j can be found in polynomial time, just by trying out all vertices in X_j. We give the configuration $\mathcal{C}_B$ as follows: We set $B_j = A_j$ for all $j \in [k]$. For each heavy set A_j such that $X_j \neq \emptyset$, we take the cascade of B_j as the cascade of A_j appended with x_j. For each heavy set A_j such that $X_j = \emptyset$, we take the cascade of B_j as the cascade of A_j. This construction of $\mathcal{C}_B$ clearly takes only polynomial time. Note that the rank of cascade vertex x_j is $\ell + 1$ and the level of all vertices in $X_j \setminus \{x_j\}$ is $\ell + 1$ in $\mathcal{C}_B$. Also, all the vertices that had level at most ℓ in $\mathcal{C}_A$ have the same level in $\mathcal{C}_B$. Conversely, all the vertices that have level at most ℓ in $\mathcal{C}_B$ had the same level in $\mathcal{C}_A$. Also, all the vertices that have level at least $\ell + 1$ in $\mathcal{C}_B$ had their level as ∞ in $\mathcal{C}_A$. Also, if a vertex was a cascade vertex with rank $\ell' \leq \ell$ in $\mathcal{C}_A$, so is it in $\mathcal{C}_B$. Conversely, if a vertex is a cascade vertex with rank $\ell' \leq \ell$ in $\mathcal{C}_B$, so was it in $\mathcal{C}_A$. Also, notice that $S_B = S_A$.

First, we claim that $\mathcal{C}_B$ is $(\ell{+}1)$-good. Suppose otherwise for the sake of contradiciton. Then, one of the 3 conditions in the definition of ℓ-good configuration should be violated.

Case 1. Condition 1 is violated, i.e., highest rank of a cascade vertex in $\mathcal{C}_B$ is not $\ell + 1$: Since all the cascade vertices that we newly introduced (i.e., the x_j's) have their rank as $\ell + 1$, there must be no new cascade vertices introduced. This implies that $X_j = \emptyset$ for each heavyset A_j. For each j such that A_j is a heavy set with a non-null cascade, let y_j be the highest ranked cascade vertex in A_j. For each j such that A_j is a heavy set with a null cascade, let y_j be t_j. Let Y be the set of all y_j such that A_j is a heavy set. Note that $|Y| \leq k - 1$ as A is an SFPP and hence has at least one light set. Hence Y is not a cutset of G as G is k-connected. Let Z_∞ be the set of all vertices in $V \setminus Y$ that have level ∞ and Z be the remaining vertices in $V \setminus Y$. Since A is an SFPP, $S_A \neq \emptyset$, and since all vertices in S_A have level ∞, we have that $Z_\infty \neq \emptyset$. Also, Z is not empty because there exists at least one light set in A and the vertices in a light set are in $V \setminus Y$. There is at least one edge between Z_∞ and Z in G, as Y is not a cutset of G. Hence, there exists an edge uv such that $u \in Z_\infty$ and $v \in Z$. If $u \in S_A$, then uv is a bridging-edge which is a contradiction by our assumption that $\mathcal{C}_A$ does not have a bridging-edge. Hence $u \in A_j$ for some $j \in [k]$. Note that A_j has to be a heavy set, as u has level ∞. We have that u is not a cascade vertex (as all cascade vertices with level ∞ are in Y) and $u \neq t_j$ (as all t_j such that level$(t_j) = \infty$ are in Y). Also, v is not of level ℓ as otherwise, $u \in X_j$ but we said that X_j is empty. But then, v has level at most $\ell - 1$, u has level ∞, and there is an edge uv. This means that v does not satisfy maximality property and hence that $\mathcal{C}_A$ is not ℓ-maximal, which is a contradiction.

Case 2. Condition 2 is violated, i.e., $\mathcal{C}_B$ is not $(\ell{+}1)$-maximal: Then, there is a vertex x of level at most ℓ that does not satisfy maximality property in $\mathcal{C}_B$. Let ℓ' be the level of x in $\mathcal{C}_B$. Let j be such that $x \in B_j$. By the definition of maximality property, x has a neighbor y such that : y has level at least $\ell' + 2$ in $\mathcal{C}_B$, y is not a rank $(\ell' + 1)$-cascade vertex in $\mathcal{C}_B$, and $y \neq t_j$.

Case 2.1. $\ell' \leq \ell - 2$: Then y has level at most ℓ in $\mathcal{C}_B$ and hence had the same level in $\mathcal{C}_A$. Thus y has level at least $\ell' + 2$ in $\mathcal{C}_A$, y is not a rank $(\ell' + 1)$-cascade vertex in $\mathcal{C}_A$, and $y \neq t_j$, implying that x did not satisfy maximality property in $\mathcal{C}_A$, implying that $\mathcal{C}_A$

is not ℓ-maximal, which is a contradiction.

Case 2.2. $\ell' = \ell - 1$: Then y has level at least $\ell + 1$ in $\mathcal{C}_B$, y is not a rank-ℓ cascade vertex in $\mathcal{C}_B$, and $y \neq t_j$, implying that y has level ∞ in $\mathcal{C}_A$, implying that x did not satisfy maximality property in $\mathcal{C}_A$, implying that $\mathcal{C}_A$ is not ℓ-maximal, which is a contradiction.

Case 2.3 $\ell' = \ell$: Then y has level at least $\ell + 2$ in $\mathcal{C}_B$, , y is not a rank-$(\ell + 1)$ cascade vertex in $\mathcal{C}_A$, $y \neq t_j$, and y is adjacent to a vertex of level ℓ. Then $y \in X_j$ by the definition of X_j. Hence either y has level $\ell + 1$ in $\mathcal{C}_B$ or y is a cascade vertex of level $\ell + 1$ in $\mathcal{C}_B$, which is a contradiction.

Case 3. Condition 3 is violated, i.e., there is a bridging-edge e from S_B to a vertex having level at most ℓ in $\mathcal{C}_B$. But then e is also a bridging-edge from S_A to a vertex having level at most ℓ in $\mathcal{C}_A$. However, $\mathcal{C}_A$ has no bridging-edges by the precondition of the lemma. Thus, we have a contradiction. Hence, $\mathcal{C}_B$ is $(\ell + 1)$-good.

It only remains to prove that $\mathcal{C}_B > \mathcal{C}_A$. All vertices that had level at most ℓ in $\mathcal{C}_A$ retained their levels in $\mathcal{C}_B$. And, at least one level-∞ vertex of $\mathcal{C}_A$ became a level-$(\ell + 1)$ vertex in $\mathcal{C}_B$ because there is a rank-$(\ell + 1)$ cascade vertex in $\mathcal{C}_B$. Since $\mathcal{C}_A$ had no level-$(\ell + 1)$ vertices, this means that $\mathcal{C}_B > \mathcal{C}_A$. □

Lemma 6.17. *Given an ℓ-good configuration $\mathcal{C}_A = (A = (A_1, \ldots, A_k), D_A)$ having a bridging-edge, we can find in polynomial time a valid configuration $\mathcal{C}_B = (B = (B_1, \ldots, B_k), D_B)$ such that one of the following holds:*

(1) $\mathcal{C}_B > \mathcal{C}_A$, and $\mathcal{C}_B$ is an ℓ-good configuration, or

(2) $\mathcal{C}_B \geq \mathcal{C}_A$, and $\mathcal{C}_B$ is an $(\ell - 1)$-good configuration with a bridging-edge.

Proof. Let uv be a bridging-edge where $u \in S_A$. Let A_{j^*} be the set containing v. Note that level$(v) = \ell$ because $\mathcal{C}_A$ is ℓ-good. We keep $B_j = A_j$ for all $j \neq j^*$. But we modify A_{j^*} to get B_{j^*} as described below. We will maintain that if A_j is a heavy set then B_j is also a heavy set for all j, and hence maintain that $L_B \leq L_A$.

Case 1: A_{j^*} is a light set (i.e., when $\ell = 0$). We take $B_{j^*} = A_{j^*} \cup \{u\}$. For all j such that B_j is a heavy set, we take cascade of B_j as the null cascade. We have $w(A_{j^*}) \leq T_j - 1$ because A_{j^*} is a light set. So, $w(B_{j^*}) = w(A_{j^*}) + w(u) \leq (T_j - 1) + w_{max}$, and hence B_{j^*} is fitted. Also, $G[B_{j^*}]$ is connected and hence $(B_1, \ldots, B_k)$ is an FPP. We have $\mathcal{C}_B >_0 \mathcal{C}_A$ because either B_{j^*} became a heavy set in which case $L_B < L_A$, or it is a light set in which case $L_B = L_A$ and $N_B^0 > N_A^0$. The configuration $\mathcal{C}_B$ is 0-good because, the highest rank of a cascade vertex is 0 as there are only null cascades, any valid configuration is 0-maximal, and any bridging-edge has to be to a level-0 vertex as there are only levels 0 and ∞ in $\mathcal{C}_B$.

Case 2: A_{j^*} is a heavy set i.e., when $\ell \geq 1$.

Case 2.1: $w(A_{j^*} \cup \{u\}) \leq T_j + w_{max} - 1$. We take $B_{j^*} = A_{j^*} \cup \{u\}$. For each j such that B_j is a heavy set (A_j is also heavy set for such j), we set the cascade of B_j the same as the cascade of A_j. $G[B_{j^*}]$ is clearly connected and B_{j^*} is fitted by assumption of the case that we are in. Hence B is indeed an FPP. Observe that all vertices that had level $\ell' \leq \ell$ in $\mathcal{C}_A$ still has level ℓ' in $\mathcal{C}_B$. Since level(v) was ℓ in $\mathcal{C}_A$ by ℓ-goodness of $\mathcal{C}_A$, u has level ℓ in $\mathcal{C}_B$; and u had level ∞ in $\mathcal{C}_A$. Hence, $\mathcal{C}_B >_\ell \mathcal{C}_A$ and thus $\mathcal{C}_B > \mathcal{C}_A$. It only remains to prove that the configuration $\mathcal{C}_B$ remains ℓ-good. Suppose $\mathcal{C}_B$ is not ℓ-good. We know that the highest rank of a cascade vertex is still ℓ as we did not change any cascades, and that any new bridging edges introduced have to be to the vertex u that is at level ℓ. Hence, for $\mathcal{C}_B$ to be not ℓ-good, it has to be not ℓ-maximal. Then, there is a vertex x of level at most $\ell - 1$ that does not satisfy maximality property in $\mathcal{C}_B$. Let ℓ' be the level of x in $\mathcal{C}_B$. Let j be such that $x \in B_j$. By the definition of maximality property, x has a neighbor y such that : y has level at least $\ell' + 2$ in $\mathcal{C}_B$, y is not a rank $(\ell' + 1)$-cascade vertex in $\mathcal{C}_B$, and $y \neq t_j$. But then y has level at least $\ell' + 2$ in $\mathcal{C}_A$, and y is not a rank $(\ell' + 1)$-cascade vertex in $\mathcal{C}_A$. This implies that x does not satisfy maximality in $\mathcal{C}_A$ also, implying that $\mathcal{C}_A$ is not ℓ-maximal, a contradiction.

Case 2.2: $w(A_{j^*} \cup \{u\}) \geq T_j + w_{max}$. Let z be the cascade vertex of rank ℓ in A_{j^*}. Note that A_{j^*} should have such a cascade vertex as $v \in A_{j^*}$ has level ℓ. Let $\bar{R}$ be $A_{j^*} \setminus (R_A(z) \cup z)$, i.e., $\bar{R}$ is the set of all vertices in $A_{j^*} \setminus \{z\}$ with level ∞. We initialize $B_{j^*} := A_{j^*} \cup \{u\}$. Now, we delete vertices one by one from B_{j^*} in a specific order until B_{j^*} becomes fitted. We choose the order of deleting vertices such that $G[B_{j^*}]$ remains connected. Consider a spanning tree τ of the connected subgraph $G[\bar{R} \cup \{z\}]$. Suppose that $\bar{R}$ is not empty. Then, the tree τ has at least 2 leaves. Thus τ has at least one leaf distinct from z. We delete this leaf from B_{j^*} and τ. We repeat this process until τ is just the single vertex z or B_{j^*} becomes fitted. If B_{j^*} is not fitted even when τ is the single vertex z, then delete z from B_{j^*}. If B_{j^*} is still not fitted then delete u from B_{j^*}. Note that at this point $B_{j^*} \subset A_{j^*}$ and hence is fitted. Also, note that $G[B_{j^*}]$ remains connected. Hence $(B_1, \ldots, B_k)$ is an FPP. B_{j^*} does not become a light set because B_j became fitted when the last vertex was deleted from it. Before this vertex was deleted, it was not fitted and hence had weight at least $T_{j^*} + w_{max}$ before this deletion. Since the last vertex deleted has weight at most w_{max}, B_{j^*} has weight at least T_{j^*} and hence is a heavy set. Now we branch into two subcases for defining the cascades.

Case 2.2.1: $z \in B_{j*}$ (i.e, z was not deleted from B_{j^*} in the process above). For each j such that B_j is a heavy set, the cascade of B_j is taken as the cascade of A_j. Since a new level-ℓ vertex u is added and all vertices that had level at most ℓ retain their level, we have that $\mathcal{C}_B >_\ell \mathcal{C}_A$ and hence $\mathcal{C}_B > \mathcal{C}_A$. Also, $\mathcal{C}_B$ remains ℓ-good, by the same arguments as that in Case 2.1.

Case 2.2.2: $z \notin B_{j^*}$ (i.e, z was deleted from B_{j^*}). For each j such that B_j is a heavy set, we set the cascade of B_j as the cascade of A_j but with the rank ℓ cascade vertex (if it has any) deleted from it. We have $\mathcal{C}_B \geq_{\ell-1} \mathcal{C}_A$ because all vertices that were at a level of $\ell' = \ell - 1$ or smaller, retain their levels. Hence, $\mathcal{C}_B \geq \mathcal{C}_A$. All the vertices that we deleted from B_{j^*} had no edges to level-$\ell - 2$ or lower level vertices in $\mathcal{C}_A$, so as to satisfy ℓ-maximality of $\mathcal{C}_A$. Hence there are no bridging-edges in $\mathcal{C}_B$ to vertices that are at a level at most $\ell - 2$, and all vertices at a level at most $\ell - 2$ still maintain the maximality property. Also, we did not introduce any cascade vertices. Hence, $\mathcal{C}_B$ is $(\ell - 1)$-good. It only remains to prove that there is a bridging-edge $u'v'$ in $\mathcal{C}_B$ such that $level(v') = \ell - 1$. We know $z \in S_B$. Since z was a rank ℓ cascade vertex in $\mathcal{C}_A$, z had an edge to z' such that z' had level $\ell - 1$ in $\mathcal{C}_A$. Observe that level of z' is $\ell - 1$ in $\mathcal{C}_B$ as well. Hence, zz' is the required bridge. □

Proof of Theorem 6.4(a): We always maintain a configuration $\mathcal{C}_A = (A, D_A)$ that is ℓ-good for some $\ell \geq 0$. We start with the 0-good configuration where $A_j = \{t_j\}$ and the cascades of all heavy sets are null cascades. If the FPP A is not an SFPP at any point, then we terminate and we have the required partition. So assume A is an SFPP.

Then, we show that we can find a new configuratoin $\mathcal{C}_B$ such that $\mathcal{C}_B > \mathcal{C}_A$ and $\mathcal{C}_B$ is ℓ'-good for some $\ell' \geq 0$, in polynomial time. We distinguish the two cases when the current configuration has a briding-edge and when it does not.

If our current configuration $\mathcal{C}_A$ is an ℓ-good configuration that has no bridging-edge, then we use Lemma 6.16 to get a configuration $\mathcal{C}_B$ such that $\mathcal{C}_B > \mathcal{C}_A$ and B is $(\ell{+}1)$-good. We take $\mathcal{C}_B$ as the new current configuration $\mathcal{C}_A$.

If our current configuration $\mathcal{C}_A$ is an ℓ-good configuration with a bridging-edge, then we apply Lemma 6.17. Then we get a configuration $\mathcal{C}_B$ such that either

(1) $\mathcal{C}_B > \mathcal{C}_A$ and $\mathcal{C}_B$ is ℓ-good, or

(2) $\mathcal{C}_B \geq \mathcal{C}_A$, $\mathcal{C}_B$ is $\ell - 1$ good , and has a bridging-edge.

If the second outcome happens, Lemma 6.17 is again applicable on $\mathcal{C}_B$. We apply the lemma repeatedly until the first outcome happens. Since the second outcome can happen only for at most ℓ times (as a (-1)-good configuration is not defined), finally the first outcome will happen, and hence we end up with an ℓ'-good configuration $\mathcal{C}_{B'}$ for some $\ell' \geq 0$ such that $\mathcal{C}_{B'} > \mathcal{C}_A$. We take $\mathcal{C}'_B$ as the new current configuration $\mathcal{C}_A$.

Thus, in either case, we get a strictly better configuration that is ℓ'-good for some $\ell' \geq 0$ in polynomial time. We call this an iteration of our algorithm. Notice that the number of iterations possible is at most the number of distinct configuration vectors possible. Hence let us estimate the number of distinct configuration vectors possible. First let us fix the first entry of the configuration vector (i.e. the number of light sets) and estimate the number of such distinct configuration vectors possible. Let N_i^ℓ be the number of distinct configuration vectors with number of light sets as i and the highest level in the configuration being exactly ℓ. Using basic combinatorics, $N_i^\ell \leq \binom{n}{\ell}$. Hence, when the highest level attained throughout the run of the algorithm is at most ℓ, the number of distinct configurations appearing during the run of the algorithm is at most $k \cdot \ell \cdot \binom{n}{\min(\ell, n/2)}$. Since the level of any vertex can be at most n, the number of iterations of our algorithm is at most $kn \cdot \binom{n}{n/2}$, which is at most $nk \cdot 2^n$. Since each iteration runs in polynomial time as guaranteed by the two lemmas, the required running time is $\mathcal{O}^*(2^n)$. □

The proof of Theorem 6.4(b) follows closely with the proof of Theorem 6.4(a), but makes use of an observation about the rank of a vertex in the local search algorithm, to give an improved bound on the number of configuration vectors navigated by the algorithm.

Proof of Theorem 6.4(b): Since any k-connected graph is also $(\lfloor k/2 \rfloor + 1)-$vertex connected, the algorithm will give the required partition due to Theorem 6.4(a). We only need to prove the better running time claimed by Theorem 6.4(b). For this, we show that the highest level attained by any vertex during the algorithm is at most $2n/(k-2)$. This means the number of distinct configuration vectors is at most $nk \cdot \binom{n}{\min(2n/(k-2), n/2)}$. For $k \geq 6$, we have $2n/(k-2) \leq n/2$. We can assume $k \geq 6$ as otherwise the $\mathcal{O}^*(2^n)$ running time already implies $\mathcal{O}^*(2^{n \log k/k})$ running time. Hence we have that the running time is $\mathcal{O}^*(\binom{n}{2n/(k-2)})$, which is $\mathcal{O}^*(2^{\mathcal{O}((n/k)\log k)})$, as claimed. Hence, it only remains to prove that the highest rank is at most $2n/(k-2)$.

For this, observe that in an ℓ-good configuration, for each $0 \leq i < \ell$, the union of all vertices having level i and the set of $(\lfloor k/2 \rfloor + 1)$ terminals together forms a cutset. Since the graph is k-connected, this means that for each $0 \leq i < \ell$, the number of vertices having level i is at least $k/2 - 1$. The required bound on the rank easily follows. □

6.3.3 Some consequences of Theorem 6.8

Proof of Corollary 6.5: Set the weight $w(v)$ of a vertex v to be its degree. Observe that the total weight is $2|E(G)|$. Since G is k-connected, each vertex has degree at least k. Thus $k|V(G)| \leq 2|E(G)|$. This implies that the maximum weight $w_{max} = \Delta(G) < |V(G)| \leq 2|E(G)|/k$. Now, we apply Theorem 6.4(a) with each T_j as $2|E(G)|/k$. (We assume for simplicity that $2|E(G)|$ is divisible by k, otherwise it can be dealt with easily). This gives the statement (a) of the corollary. Similarly the statement (b) follows by applying Theorem 6.4(b) with each T_j as $2|E(G)|/(k/2)$. □

Proof of Corollary 6.6: Let $n = |V|$ and $m = |E|$. Set the weight $w(v)$ of a vertex v to be $(cm/n) + \mathsf{deg}(v)$. Observe that the total weight is $(c+2)m$. As in the proof of previous corollary, we have $kn \leq 2m$. This implies that the maximum weight $w_{max} = cm/n + \Delta(G) < cm/n + 2m/k \leq (c+2)m/k$. Now, applying Theorem 6.4(a) with each T_j as $(c+2)m/k$ gives a connected partition $\bigcup_{j=1}^{k} V_j$ where for each j, we have $w(V_j) \leq (T_j + w_{max}) \leq (2c+4)m/k$. From this we get that $\mathsf{deg}(V_j) + (cm/n)|V_j| \leq (2c+4)m/k$. This implies $\mathsf{deg}(V_j) \leq (2c+4)m/k$ and $|V_j| \leq \frac{2c+4}{c}(n/k)$. □

6.4 Upper Bounds for Spanning Tree Congestion

In this section, we prove Theorems 6.7, 6.8 and 6.9. The section is organized as follows. In subsection 6.4.1, we prove Theorem 6.7 regarding the STC of k-connected graphs. In subsection 6.4.2, we give an algorithmic framework which is used to prove both the Theorems 6.8 and 6.9. For proving Theorem 6.9, we make use of an algorithm for *confluent flows* by Chen et al. [46]. We review confluent flows and the algorithm by Chen et al. in subsection 6.4.3. In subsection 6.4.4, we complete the proof of Theorem 6.9. Finally, in subsection 6.4.4, we give the proof of Theorem 6.8.

6.4.1 STC of k-connected graphs

Here, we prove Theorem 6.7. Towards this, let us first prove the following lemma. See Figure 6.3 for an illustration of the lemma.

Lemma 6.18. *In a graph $G = (V, E)$, let t_1 be a vertex, and let $t_2, \cdots, t_\ell$ be any $(\ell - 1)$ neighbors of t_1. Suppose that there exists a connected ℓ-partition $\cup_{j=1}^{\ell} V_\ell$ of V such that for all $j \in \ell$, $t_j \in V_j$, and the sum of degree of vertices in each V_j is at most D. Let τ_j be an arbitrary spanning tree of $G[V_j]$. Let e_j denote the edge $t_1 t_j$. Let τ be the spanning tree of G defined as $\tau := \left(\cup_{j=1}^{\ell} \tau_j\right) \bigcup \left(\cup_{j=2}^{\ell} e_j\right)$. Then τ has congestion at most D.*

Proof. Observe that for any edge e in τ, one of the two sets of the cut induced by e in τ is a subset of some V_j. Since the total number of outgoing edges from V_j is at most D, the number of edges in the cut is at most D. Thus we have that the congestion of any edge in τ is at most D. □

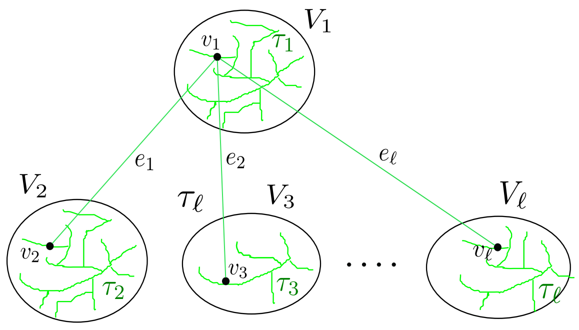


Figure 6.3: An illustration of Lemma 6.18.

Proof of Theorem 6.7: Follows directly from Corollary 6.5 and the above lemma. □

6.4.2 Framework for the algorithms for STC

It remains to prove Theorems 6.8 and 6.9. We give two algorithms, one for Theorem 6.8 and the other for Theorem 6.9. We give a single framework that works for both the algorithms. The framework is described below in Algorithm 3 and illustrated by Figures 6.4 and 6.5. It is a recursive algorithm; the parameters $\hat{m}, \hat{n}$ are global parameters, where $\hat{m}$ is the number of edges in the input graph G in the first level of the recursion; and $\hat{n}$ is the number of vertices in G. The parameters n_H, m_H are local parameters, where m_H is the number of edges in the input graph H in the current level of the recursion; and n_H is the number of vertices in H. The only difference between the two algorithms is in Line 15, on how this line is executed, with trade-off between the running time $T(\hat{m}, n_H, m_H)$ of the step, and the congestion guarantee $D(\hat{m}, n_H, m_H)$. For proving Theorem 6.8, we use the algorithm given by Theorem 6.7, yielding $D(\hat{m}, n_H, m_H) \le 8m_H\sqrt{n_H/\hat{m}}$ and $T(\hat{m}, n_H, m_H) = \mathcal{O}^*\left(2^{\mathcal{O}\left(n_H \log n_H / \sqrt{\hat{m}/n_H}\right)}\right)$. For proving Theorem 6.9, we make use of an algorithm for confluent flow by Chen et al. [46], which yields $D(\hat{m}, n_H, m_H) \le 16m_H\sqrt{n_H/\hat{m}} \cdot \log n_H$ and $T(\hat{m}, n_H, m_H) = \text{poly}(n_H, m_H)$.

6.4.3 Review of algorithm for confluent flows by Chen et al.

In this section, we discuss about confluent flows and the algorithm by Chen et al. [46]. We also prove a corollary about connected k-partition and spanning tree congestion based on their algorithm.

Single-Commodity Confluent Flow In a *single-commodity confluent flow* problem, the input includes a graph $G = (V, E)$, a *demand* function $w : V \to \mathbb{R}_{>0}$ and ℓ sinks $t_1, \cdots, t_\ell \in V$. For each $v \in V$, a flow of amount $w(v)$ is routed from v to one of the sinks. But there is a restriction: at every vertex $u \in V$, the outgoing flow must leave u on at most 1 edge, i.e., the outgoing flow from u is unsplittable. The problem is to seek a flow satisfying the demands which minimizes the *node congestion*, i.e., the maximum incoming flow among all vertices. Since the incoming flow is maximum at one of the sinks, it is equivalent to minimize the maximum flow received among all sinks. (Here, we assume that no flow entering a sink will leave.)

Algorithm 3: `FindLCST`$(H, \hat{m})$

```
   Input  : A connected graph $H = (V_H, E_H)$ on $n_H$ vertices and $m_H$ edges
   Output : A spanning tree $\tau$ of $H$
1  if $m_H \leq 8\sqrt{\hat{m} n_H}$ then
2      return an arbitrary spanning tree of $H$
3  end
4  $k \leftarrow \left\lceil \left( \sqrt{\hat{m}/n_H} \right) \right\rceil$
5  $Y \leftarrow$ a global minimum vertex cut of $H$
6  if $|Y| < k$ then
7      $X \leftarrow$ the smallest connected component in $H[V_H \setminus Y]$ (See Figure 6.4)
8      $Z \leftarrow V_H \setminus (X \cup Y)$
9      $\tau_1 \leftarrow$ FindLCST( $H[X], \hat{m}$ )
10     $\tau_2 \leftarrow$ FindLCST( $H[Y \cup Z], \hat{m}$); ($H[Y \cup Z]$ is connected as $Y$ is a global min
       cut)
11     return $\tau := \tau_1 \cup \tau_2 \cup$ (an arbitrary edge between $X$ and $Y$)
12 else
13     $t_1 \leftarrow$ an arbitrary vertex in $V_H$
14     Pick $\lfloor k/2 \rfloor$ neighbors of $t_1$ in the graph $H$; denote them by $t_2, t_3, \cdots, t_{\lfloor k/2 \rfloor + 1}$.
       Let $e_j$ denote edge $t_1 t_j$ for $2 \leq j \leq \lfloor k/2 \rfloor + 1$. (See Figure 6.5)
15     Compute a connected $(\lfloor k/2 \rfloor + 1)$-partition of $H$, denoted by $\cup_{j=1}^{\lfloor k/2 \rfloor + 1} V_j$, such
       that for each $j \in [\lfloor k/2 \rfloor + 1]$, $t_j \in V_j$, and the total degree (w.r.t. graph $H$) of
       vertices in each $V_j$ is at most $D(\hat{m}, n_H, m_H)$. Let the time needed be
       $T(\hat{m}, n_H, m_H)$.
16     For each $j \in [\lfloor k/2 \rfloor + 1]$, $\tau_j \leftarrow$ an arbitrary spanning tree of $G[V_j]$
17     return $\tau := \left( \cup_{j=1}^{\lfloor k/2 \rfloor + 1} \tau_j \right) \cup \left( \cup_{j=2}^{\lfloor k/2 \rfloor + 1} e_j \right)$
18 end
```

Single-commodity splittable flow problem is almost identical to single-commodity confluent flow problem, except that the above restriction is dropped, i.e., now the outgoing flow at u can split along multiple edges. Note that here, the maximum incoming flow might not be at a sink. It is well known that single-commodity splittable flow can be solved in polynomial time. For brevity, we drop the phrase "single-commodity" from now on. Chen et. al. proved the following theorem:

Theorem 6.19 ([46, Section 4]). *Suppose that given graph G, demand w and ℓ sinks, there is a splittable flow with node congestion q. Then there exists a polynomial time algorithm which computes a confluent flow with node congestion at most $(1 + \ln \ell)q$ for the same input.*

We now prove a corollary that follows from Theorem 6.19 and Corollary 6.5.

Corollary 6.20. *Let G be a k-connected graph with m edges. Then for any $\ell \leq k$ and for any ℓ vertices $t_1, \cdots, t_\ell \in V$, there exists a polynomial time algorithm which computes an connected ℓ-partition $\cup_{j=1}^{\ell} V_\ell$ such that for all $j \in \ell$, $t_j \in V_j$, and the total degrees of*

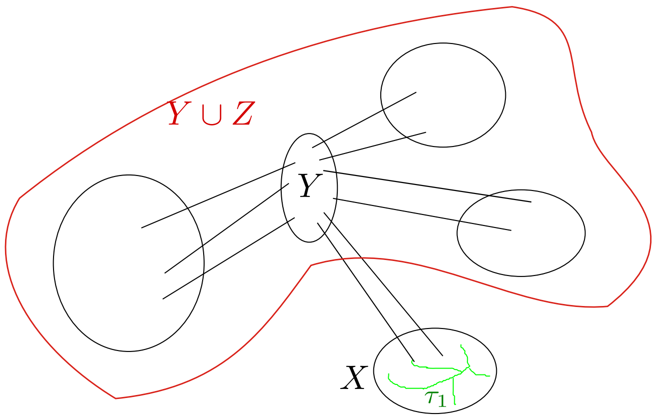


Figure 6.4: The scenario in Algorithm 3 when the graph has low connectivity. The vertex set Y is a global minimum vertex cut of the graph. The vertex set X is the smallest connected component after the removal of Y, and Z is the union of all the other connected components.

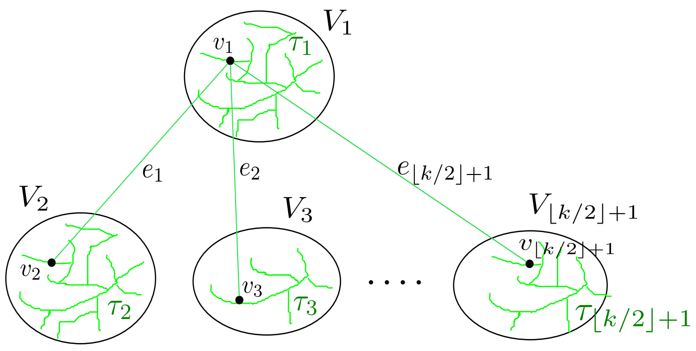

V_1
v_1
$\mathcal{T}_1$
e_1
e_2
$e_{\lfloor k/2 \rfloor+1}$
V_2
V_3
$V_{\lfloor k/2 \rfloor+1}$
v_2
v_3
$v_{\lfloor k/2 \rfloor+1}$
$\mathcal{T}_2$
$\mathcal{T}_3$
$\mathcal{T}_{\lfloor k/2 \rfloor+1}$
....

Figure 6.5: The scenario in Algorithm 3 when the graph has high connectivity.

vertices in each V_j is at most $4(1 + \ln \ell)m/\ell$. Moreover, we can find a spanning tree of G with congestion at most $4(1 + \ln \ell)m/\ell$ in polynomial time.

Proof. First of all, we set the demand of each vertex in the flow problem to be the the degree of the vertex in G, and $t_1, \cdots, t_\ell$ as the sinks in the flow problem.

By Corollary 6.5, there exists an connected ℓ-partition $\cup_{j=1}^{\ell} U_\ell$ such that for all $j \in [\ell]$, $t_j \in U_j$, and the total degrees of vertices in each U_j is at most $4m/\ell$. With this, by routing the demand of a vertex in U_j to t_j via an arbitrary path in $G[U_j]$ only, we construct a splittable flow with node congestion at most $4m/\ell$. By Theorem 6.19, one can construct a confluent flow with node congestion at most $4(1+\ln \ell)m/\ell$ in polynomial time. In the confluent flow, all the flow originating from one vertex goes completely into one sink. Set V_j to be the set of vertices such that the flows originating from these vertices go into t_j. It then follows that $\cup_{j=1}^{\ell} V_\ell$ is our desired connected ℓ-partition. Then, by applying Lemma 6.18, we get the required spanning tree in polynomial time. □

6.4.4 Proof of Theorem 6.9

The algorithm specified in Theorem 6.9 is obtained from the framework in Algorithm 3 by using the algorithm given by Corollary 6.20 to compute the connected partition in Line 15 of Algorithm 3. We prove below that this algorithm satisfies the congestion bound and the running time guarantee given in Theorem 6.9.

Correctness and Congestion Analysis. We view the whole recursion process as a recursion tree. There is no endless loop, since down every path in the recursion tree, the number of vertices in the input graphs are *strictly* decreasing. Note that the leaf of the recursion tree is resulted either by (i) Line 2, i.e., if the input graph H to that call satisfies $m_H \leq 8\sqrt{\hat{m}n_H}$, (ii) or by Line 13, i.e, if connectivity of H is at least k. An internal node appears only when the connectivity of the input graph H is lesser than k, and it makes two recursion calls in that case.

We prove the following statement by induction from bottom-up: for each graph which is the input to some call in the recursion tree, the returned spanning tree of that call is indeed a spanning tree and has congestion at most $16\sqrt{\hat{m}n_H}\log n_H$.

We first handle the two basis cases (i) and (ii). In case (i), `FindLCST` returns an arbitrary spanning tree, and the congestion is bounded by $m_H \leq 8\sqrt{\hat{m}n_H}$. In case (ii), by Corollary 6.20, `FindLCST` returns a spanning tree with congestion at most

$$4(1+\ln(k/2))m_H/(k/2) \leq 16m_H\sqrt{n_H/\hat{m}}\cdot\log n_H \leq 16\sqrt{\hat{m}n_H}\cdot\log n_H.$$

Next, let H be the input graph to a call which is represented by an internal node of the recursion tree. Recall the definitions of X, Y, Z, τ_1, τ_2 in the algorithm. Note that the connectedness of $H[Y\cup Z]$ follows from the minimality of cutset Y. It is then easy to see that τ is indeed a spanning tree in this case.

Let $|X| = x$. Note that $1 \leq x \leq n_H/2$. Then, the congestion of the returned spanning tree is at most

$$\begin{aligned} &\max\{\text{ congestion of } \tau_1 \text{ in } H[X]\text{ , congestion of } \tau_2 \text{ in } H[Y\cup Z]\ \} \ + \ |X|\cdot|Y| \\ &\leq 16\sqrt{\hat{m}(n_H - x)}\log(n_H - x) \ + \ \left(\sqrt{\hat{m}/n_H} + 1\right)\cdot x. \end{aligned} \tag{6.1}$$

Viewing x as a real variable, by taking derivative, it is easy to see that the above expression is maximized at $x = 1$. Thus the congestion is at most

$$16\sqrt{\hat{m}(n_H-1)}\log(n_H-1) + \sqrt{\hat{m}/n_H} + 1 \quad \leq \quad 16\sqrt{\hat{m}n_H}\log n_H.$$

Runtime Analysis. At every internal node of the recursion tree, the algorithm makes two recursive calls with two vertex-disjoint and strictly smaller (w.r.t. vertex size) inputs. The dominating knitting cost is in Line 5 for computing a global minimum vertex cut, which is well-known that it can be done in polynomial time. Since at every leaf of the recursion tree the running time is polynomial, by standard analysis on divide-and-conquer algorithms, the running time of the whole algorithm is polynomial, which completes the proof of Theorem 6.9.

6.4.5 Proof of Theorem 6.8

The algorithm specified in Theorem 6.8 is obtained from the framework in Algorithm 3 by using the algorithm given by Theorem 6.7 to compute the connected partition in Line 15 of Algorithm 3. We now prove that this algorithm satisfies the congestion bound and the running time guarantee given in Theorem 6.9. Instead of giving the full proof, we only point out the differences from the proof of Theorem 6.9.

First, in handling the basis case (ii), by Theorem 6.7, we have an improved upper bound on the congestion of the returned tree, which is $8m_H/\sqrt{\hat{m}/n_H} \leq 8\sqrt{\hat{m}n_H}$. Thus, (6.1) can be improved to

$$8\sqrt{\hat{m}(n_H - x)} \;+\; \sqrt{\frac{\hat{m}}{n_H}} \cdot x.$$

Again, by viewing x as a real variable and taking derivative, it is easy to see that the above expression is maximized at $x = 1$. So the above bound is at most

$$8\sqrt{\hat{m}(n_H - 1)} \;+\; \sqrt{\frac{\hat{m}}{n_H}} \;\leq\; 8\sqrt{\hat{m}n_H}, \text{ as desired.}$$

Concerning the running time, it is clear that in the worst case, it is dominated by the calls to the algorithm in Theorem 6.4(b). Note that the number of such calls is at most $\hat{n}$, since each call to the algorithm is on a disjoint set of vertices. Hence, the required running time bound follows.

6.5 Lower Bound for Spanning Tree Congestion

Here, we give a lower bound on spanning tree congestion which matches our upper bound by proving Theorem 6.10.

We start with the following lemma, which states that for a random graph $\mathcal{G}(n,p)$, when p is sufficiently large, its *edge expansion* is $\Theta(np)$ with high probability. The proof of the lemma uses only fairly standard probability arguments and is deferred to Section 6.5.1.

Lemma 6.21. *For any integer $n \geq 4$ and $1 \geq p \geq 32 \cdot \frac{\log n}{n}$, let $\mathcal{G}(n,p)$ denote the random graph with n vertices, in which each edge occurs independently with probability p. Then with probability at least $1 - \mathcal{O}(1/n)$,*
(i) the random graph is connected,
(ii) the number of edges in the random graph is between $pn^2/4$ and pn^2, and
(iii) for each subset of vertices S with $|S| \leq n/2$, the number of edges leaving S is at least $\frac{p}{2} \cdot |S| \cdot (n - |S|)$.

In particular, for any sufficiently large integer n, when $n^2/2 \geq m \geq 16n \log n$, by setting $p = 2m/n^2$, there exists a connected graph with n vertices and $[m/2, 2m]$ edges, such that for each subset of vertices S with $|S| \leq n/2$, the number of edges leaving S is at least $\frac{m}{2n} \cdot |S| = \Theta(m/n) \cdot |S|$. We denote such a graph by $H(n,m)$.

We discuss our construction here (see Figure 6.6) before delving into the proof. The vertex set V is the union of three vertex subsets V_1, V_2, V_3, such that $|V_1| = |V_2| = |V_3| = n$, $|V_1 \cap V_2| = |V_2 \cap V_3| = \sqrt{m/n}$, and V_1, V_3 are disjoint. In each of V_1, V_2 and V_3, we

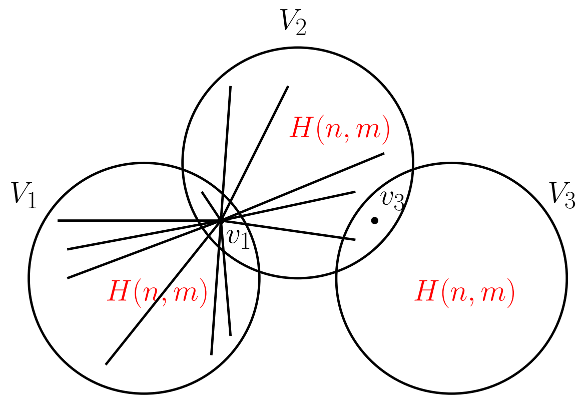


Figure 6.6: Our lower-bound construction for spanning tree congestion. V_1, V_2, V_3 are three vertex subsets of the same size. In each of the subsets, we embed expander $H(n,m)$. There is a small overlap between V_2 and V_1, V_3, while V_1, V_3 are disjoint. For any vertex $v_1 \in V_1 \cap V_2$, we add edges between it and any other vertex in $V_1 \cup V_2$; similarly, for any vertex $v_3 \in V_3 \cap V_2$, we add edges (not shown in figure) between it and any other vertex in $V_3 \cup V_2$.

embed $H(n,m)$. The edge sets are denoted E_1, E_2, E_3 respectively. Up to this point, the construction is similar to that of Ostrovskii [132], except that we use $H(n,m)$ instead of a complete graph.

The new component in our construction is adding the following edges. For each vertex $v \in V_1 \cap V_2$, add an edge between v and every vertex in $(V_1 \cup V_2) \setminus \{v\}$. The set of these edges are denoted F_1. Similarly, for each vertex $v \in V_3 \cap V_2$, add an edge between v and every vertex in $(V_3 \cup V_2) \setminus \{v\}$. The set of these edges are denoted F_3.

Proof of Theorem 6.10: Let $G = (V,E)$ be the graph constructed as above. The whole graph has $3n - 2\sqrt{m/n}$ vertices. The number of edges is at least m (due to edges in E_1 and E_3), and is at most $6m + 2\sqrt{m/n} \cdot 2n = 6m + 4\sqrt{mn}$, which is at most $7m$ for $m \geq 16n$.

It is well known that for any tree on n vertices, there exists a vertex x called a **centroid** of the tree such that, removing x decomposes the tree into connected components, each of size at most $n/2$. Now, consider any spanning tree τ of the given graph, let u be a centroid of τ. Without loss of generality, we can assume that $u \notin V_1$; otherwise we swap the roles of V_1 and V_3. Removing u (and its adjacent edges) from τ decomposes the tree τ into a number of connected components. Any of these components that intersect V_1 must contain at least one vertex of $V_1 \cap V_2$; thus the number of components intersecting V_1 is at most $\sqrt{m/n}$, and hence there exists one of them, denoted by U_j, such that

$$b_1 \;:=\; |U \cap V_1| \;\geq\; n/(\sqrt{m/n}) \;=\; n\sqrt{n/m}.$$

Let e denote the tree-edge that connects u to U. We distinguish the following three cases:

Case 1: $n\sqrt{n/m} \;\leq\; b_1 \;\leq\; n - n\sqrt{n/m}$**.** Due to the property of $H(n,m)$, the congestion of e is at least $\Theta(m/n) \cdot \min\{b_1, n-b_1\} \;\geq\; \Theta(\sqrt{mn})$.

Case 2: $b_1 > n - n\sqrt{n/m}$ and $|U \cap V_1 \cap V_2| \le \frac{1}{2} \cdot \sqrt{m/n}$. Let $W := (V_1 \cap V_2) \setminus U$. Note that by this case's assumption, $|W| \ge \frac{1}{2} \cdot \sqrt{m/n}$. Due to the edge subset F_1, the congestion of e is at least

$$\left|F_1(W\ ,\ V_1 \setminus W)\right| \ \ge\ \left(\frac{1}{2} \cdot \sqrt{m/n}\right) \cdot \frac{n}{2} \ =\ \Theta\left(\sqrt{mn}\right).$$

Case 3: $b_1 > n - n\sqrt{n/m}$ and $|U \cap V_1 \cap V_2| > \frac{1}{2} \cdot \sqrt{m/n}$. Let $W' := U \cap V_1 \cap V_2$, and let $Z := (V_2 \setminus V_1) \cap U$.

Note that $b_1 > n - n\sqrt{n/m} \ge 9n/10$. Suppose $|Z| \ge 6n/10$, then $|U| > 9n/10 + 6n/10 > |V|/2$, a contradiction to the assumption that u is a centroid. Thus, $|Z| < 6n/10$. Due to the edge subset F_1, the congestion of e is at least

$$\begin{aligned}\left|F_1(W',\ V_2 \setminus (W' \cup Z))\right| \ &\ge\ |W'| \cdot (n - |W'| - |Z|) \\ &\ge\ \left(\frac{1}{2} \cdot \sqrt{m/n}\right) \cdot \left(n - \sqrt{m/n} - \frac{6n}{10}\right) \ =\ \Theta(\sqrt{mn}). \qquad \square\end{aligned}$$

6.5.1 Proof of Lemma 6.21

It is well known that the requirements (i) and (ii) are satisfied with probability $1 - o(1/n)$. [25] For each subset S with $|S| \le n/2$, by the Chernoff bound,

$$\mathbb{P}\left[\ \left|E(S, V \setminus S)\right| \ \le\ \frac{p}{2} \cdot |S| \cdot (n - |S|)\ \right] \ \le\ e^{-p|S|(n-|S|)/8} \ \le\ e^{-pn|S|/16}.$$

Since $p \ge 32 \cdot \frac{\log n}{n}$, the above probability is at most $n^{-2|S|}$. Then by a union bound, the probability that (iii) is not satisfied is at most

$$\sum_{s=1}^{\lfloor n/2 \rfloor} \binom{n}{s} \cdot n^{-2s} \ \le\ \sum_{s=1}^{\lfloor n/2 \rfloor} n^s \cdot n^{-2s} \ \le\ \sum_{s=1}^{\lfloor n/2 \rfloor} n^{-s} \ \le\ \frac{2}{n}.$$

6.6 STC of Graphs with Expanding Properties

Recall the following definition of a (n, s, d_1, d_2, d_3, t)-expanding graph:

Definition 6.11. *A graph $G = (V, E)$ on n vertices is an (n, s, d_1, d_2, d_3, t)-expanding graph if the following four conditions are satisfied:*

(1) for each $S \subseteq V(G)$ such that $|S| = s$, $|N(S)| \ge d_1 n$;

(2) for each $S \subseteq V(G)$ such that $|S| \le s$, $|N(S)| \ge d_2|S|$;

(3) for each $S \subseteq V(G)$ such that $|S| \le n/2$ and for any $S' \subset S$, $|N_{V \setminus S}(S')| \ \ge\ |S'| - t$.

(4) For each $S \subseteq V(G)$, $|E(S, V \setminus S)| \le d_3|S|$.

Whenever we say condition (1), (2), (3) or (4) in this section, we mean the corresponding condition in the above definition. We will now prove Theorem 6.12 by presenting a polynomial time algorithm for finding a spanning tree of an (n, s, d_1, d_2, d_3, t)-expanding graph that has congestion at most

$$d_3 \cdot \left[4 \cdot \max\left\{ s+1\ ,\ \left\lceil \frac{3d_1 n}{d_2} \right\rceil \right\} \cdot \left(\frac{1}{2d_1} \right)^{\log_{(2-\delta)} 2} + t \right], \quad \text{where } \delta = \frac{t}{d_1 n}.$$

Algorithm. Let G be an (n, s, d_1, d_2, d_3, t)-expanding graph. By Condition (2) in Definition 6.11, every vertex has degree at least d_2. Let v_0 be a vertex of degree $d \geq d_2$, and let $v_1, \cdots, v_d$ be its d neighbors. We maintain a tree T rooted at v_0 such that $T = T_1 \cup T_2 \cup \cdots \cup T_d \cup \{v_0v_1, v_0v_2, \ldots, v_0v_d\}$ where $T_1, T_2, \cdots, T_d$ are trees rooted at $v_1, v_2, \ldots, v_d$ respectively. We call the $T_i' s$ as *branches*. (See Figure 6.7). We start with each branch $T_i = v_i$. In order to minimize congestion, we grow T in a balanced way, i.e., we maintain that the T_i's are roughly of the same size. A branch is called *saturated* if it contains at least $\max\left\{s+1\ ,\ \frac{3d_1 n}{d_2}\right\}$ vertices.

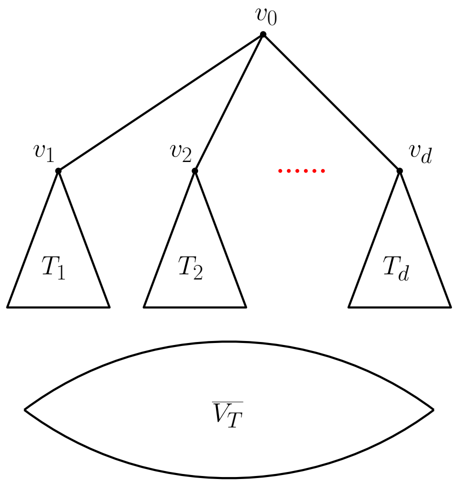


Figure 6.7: The tree T and its branches

At any point of time, let V_T be the set of vertices in T and $\bar{V_T}$ be the vertices not in T. Often, we will move a subtree of a saturated branch T_i to an unsaturated branch T_j to ensure balance. For any $x \in V_T$, let T_x denote the subtree of T rooted at x. A vertex x of a saturated branch T_i is called *transferable* (to branch T_j) if x has a neighbor y in T_j and the tree $T_j \cup \{xy\} \cup T_x$ is unsaturated. (See Figure 6.8.)

The algorithm is divided into two phases which are described below. Throughout the algorithm, whenever a branch T_i gets modified, T gets modified accordingly, and whenever T gets modified V_T and $\bar{V_T}$ gets modified accordingly.

Phase 1: Repeatedly do one of the following two actions, until $|V_T| \geq d_1 n$:
(We will prove that the precondition of at least one of the actions is satisfied if $|V_T| < d_1 n$)

(1) If there exists a $b \in \bar{V_T}$ such that b has a neighbor a in some unsaturated branch T_i:
Add the vertex b and the edge ab to branch T_i.

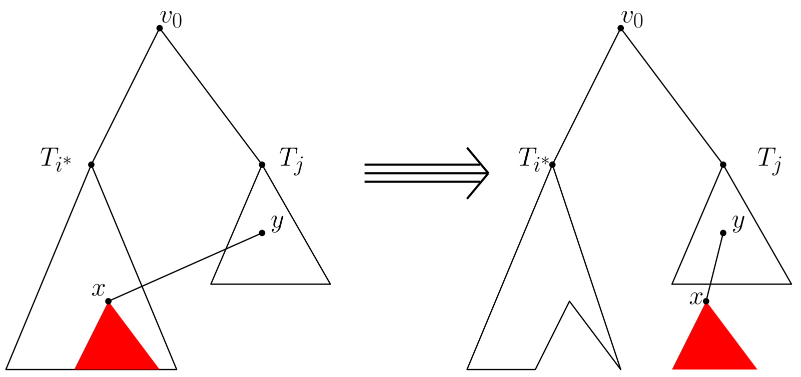


Figure 6.8: Transfer of a subtree from a saturated branch to an unsaturated branch

(2) If there exists at least one transferable vertex: (see Figure 6.8)
Find the transferable vertex x such that T_x is the smallest. Let T_{i^*} be the branch currently containing x, T_j be a branch to which it is transferable, and y be an arbitrarily chosen neighbor of x in T_j.

(a) Remove the subtree T_x from T_{i^*} and add it to T_j with x as a child of y.

(b) Pick a $b \in \bar{V_T}$ that has a neighbor a (arbitrarily chosen, if many) either in T_{i^*} or in T_j. (We will show in the analysis that such b exists). We add vertex b and edge ab to the branch containing a (i.e. to T_{i^*} or T_j).

Phase 2: While $\bar{V_T} \neq \emptyset$, repeat:
Find a maximum matching of $G[V_T, \bar{V_T}]$, the bipartite graph formed by edges of G between V_T and $\bar{V_T}$. Let M be the matching. Add all edges of M to T.

In the analysis below, we say that a tree is *saturated* if it contains at least $\mathcal{A}$ vertices; we will determine the appropriate value of $\mathcal{A}$ by the end of the analysis.

Analysis of Phase 1. We claim that during Phase 1, i.e. if $|V_T| < d_1 n$, the precondition of either step 1 or step 2 is satisfied. We also show the existence of a vertex b as specified in step 2b, whenever step 2b is reached. Given these and the fact that a vertex in $\bar{V_T}$ is moved to V_T (either in step 1 or in step 2b) during each round of Phase 1, we have that Phase 1 runs correctly and terminates after a linear number of rounds.

During Phase 1, we will also maintain the invariant that each branch has at most $\mathcal{A}$ vertices; thus, each saturated branch has exactly $\mathcal{A}$ vertices. We call this invariant the *balancedness*. Note that balancedness is not violated due to step 1, as the new vertex is added to an unsaturated branch. It is not violated during step 2 as the branches T_{i^*} and T_j (as defined in step 2) become unsaturated at the end of the step.

We define the *hidden vertices* of T (denoted by $H \equiv H_T$) as follows: they are the vertices which are not adjacent to any vertices outside the tree, i.e., to any vertex in $\bar{V_T}$. If there is an unsaturated branch with a non-hidden vertex, clearly the precondition of step 1 is satisfied. So, let us assume that all the vertices in all unsaturated branches are hidden. In such a case, we show that the precondition of step 2 is satisfied if $|V_T| < d_1 n$.

We argue that in this case $|H| \leq s$: otherwise, take a subset $H' \subset H$ of cardinality s, then by condition (2), $N(H')$, which is contained in V_T, has cardinality at least $d_1 n$. This is a contradiction as $|V_T| < d_1 n$ during Phase 1.

Since $|V_T| < d_1 n$, the number of saturated branches is at most $d_1 n/\mathcal{A}$. To ensure that at least one unsaturated branch exists, we set $\mathcal{A}$ such that $d_1 n/\mathcal{A} < d_2$. Let U denote the set of vertices in all unsaturated branches. Since all vertices in U are hidden vertices, $|U| \leq s$. Then by condition (2), $|N(U)| \geq d_2|U|$. Note that the vertices in $N(U)$ are all in the saturated branches. By the pigeon-hole principle, there exists a saturated branch containing at least

$$N(U)/(d_1 n/\mathcal{A}) \;\geq\; \frac{\mathcal{A} d_2 |U|}{d_1 n}$$

vertices of $N(U)$. By setting $\mathcal{A} \geq \frac{3d_1 n}{d_2}$, the above calculation guarantees the existence of a saturated branch containing at least $3|U| \geq |U| + 2$ vertices of $N(U)$; let T_i be such a branch.

In T_i, pick a vertex $x \in T_i \cap N(U)$ such that T_x does not contain any vertex in $N(U)$, except x. Then the size of T_x is at most $\mathcal{A} - |N(U) \cap T_i| + 1 \leq \mathcal{A} - (|U| + 1)$. Let $y \in U$ be a vertex which is adjacent to x and T_j be the branch containing y. Since T_j has at most $|U|$ vertices, x is a transferable vertex (to T_j). Thus precondition of step 2 is satisfied.

We further set $\mathcal{A} > s$ so that in each saturated branch, there is at least one unhidden vertex. In particular, T_i has an unhidden vertex, which is adjacent to some $b \in \bar{V_T}$. The vertex b is either adjacent to a vertex in T_x, or a vertex in $T_i \setminus T_x$ as required in step 2b.

Analysis of Phase 2. Since G is connected, we have that M is non-empty in each iteration of Phase 2, and hence Phase 2 terminates in linear number of rounds. At the end of Phase 2, since $\bar{V_T}$ is empty, T is clearly a spanning tree. It only remains to estimate the congestion of this spanning tree. Towards this, we state the following *modified Hall's theorem*, which is an easy corollary of the standard Hall's theorem.

Lemma 6.22. *Consider a bipartite graph H with $|L(H)| \leq |R(H)|$. Suppose that there exist $t \geq 0$ such that for any $W \subseteq L(H)$, we have $|N(W)| \geq |W| - t$. Then the bipartite graph admits a matching of size at least $|L(H)| - t$.*

Recall that Phase 2 consists of multiple rounds of finding a matching between V_T and $\bar{V_T}$. As long as $|V_T| \leq n/2$, condition (3) (with $S = V_T$) plus the modified Hall's theorem (with $L = V_T$ and $R = \bar{V_T}$) guarantees that in each round, at least $|V_T| - t$ number of vertices in V_T are matched. We now define δ as:

$$\delta := t/(d_1 n).$$

Then,

$$(1 - \delta)|V_T| = \left(1 - \frac{t}{d_1 n}\right) \cdot |V_T| \leq |V_T| - t$$

where the last inequality uses that $|V_T| \geq d_1 n$ during Phase 2. Thus at least $(1 - \delta)|V_T|$ vertices from V_T are matched in each round. Thus, after at most $\left\lceil \log_{(2-\delta)} \frac{1}{2d_1} \right\rceil$ rounds of matching, $|V_T| \geq n/2$. After reaching $|V_T| \geq n/2$, condition (3) (with $S = \bar{V_T}$) plus the modified Hall's theorem (with $L = \bar{V_T}$ and $R = V_T$) guarantees that after one more round of matching, all but t vertices are left in $\bar{V_T}$.

By the end of Phase 1, each branch had at most $\mathcal{A}$ vertices. After each round of matching, the cardinality of each branch is doubled at most. Thus, the maximum possible number of vertices in each branch after running the whole algorithm is at most

$$\mathcal{A} \cdot 2^{\left\lceil \log_{(2-\delta)} \frac{1}{2d_1} \right\rceil + 1} + t \quad \leq \quad 4\mathcal{A} \cdot \left(\frac{1}{2d_1} \right)^{\log_{(2-\delta)} 2} + t.$$

and hence the STC is at most

$$d_3 \cdot \left[4\mathcal{A} \cdot \left(\frac{1}{2d_1} \right)^{\log_{(2-\delta)} 2} + t \right].$$

Recall that we need $\mathcal{A}$ to satisfy $d_1 n/\mathcal{A} < d_2$, $\mathcal{A} \geq \frac{3d_1 n}{d_2}$ and $\mathcal{A} > s$. Thus we set $\mathcal{A} := \max\left\{ s+1 \; , \; \left\lceil \frac{3d_1 n}{d_2} \right\rceil \right\}$.

6.7 STC of Random Graphs

In this section, we prove that a random graph has an STC of $\Theta(n)$ with high probability. We first present a simple non-constructive proof that a random graph has an STC of $\mathcal{O}(n)$ with high probability, in Theorem 6.23. The proof of Theorem 6.23 uses the upper bound given by Theorem 6.7 on the STC of k-connected graphs, and the fact that for random graphs, vertex-connectivity and minimum degree are equal with high probability. Theorem 6.23 does not give an efficient algorithm; that will be given in Section 6.7.1.

Theorem 6.23. *If $G \in \mathcal{G}(n,p)$ where $p \geq 8 \log n/n$, then the spanning tree congestion of G is at most $16n$ with probability at least $1 - o(1/n)$.*

Proof. It is known that the threshold probability for a random graph being k-connected is same as the threshold probability for it having minimum degree at least k [27]. Since $p \geq 8 \log n/n$, using Chernoff bound and taking union bound over all vertices gives that G has minimum degree at least $np/2$ with probability at least $1 - o(1/n)$. Hence G is $(np/2)$-connected with probability at least $1 - o(1/n)$. We also have that the number of edges in G is at most $2n^2 p$ with probability at least $1 - o(1/n)$. Now, by using Theorem 6.7, we have that with probability at least $1 - o(1/n)$, the spanning tree congestion is at most $16n$. □

Next, we give a lower bound of $\Omega(n)$ on the STC of random graphs by proving Theorem 6.14.

Proof of Theorem 6.14: By using Chernoff Bounds and applying union bound, it is easy to show that with probability $1 - o(1/n)$, every vertex of G has degree at most $c_1 np$ for a sufficiently large constant c_1. Also, by Lemma 6.21, with probability $1 - \mathcal{O}(1/n)$, properties (i) and (iii) of that lemma holds. The rest of the proof is conditioned on the above mentioned highly probable events.

Take a spanning tree T of G which gives the minimum congestion. Let u be a centroid of the tree T, i.e., each connected component of $T \setminus \{u\}$ has at most $n/2$ vertices. If there is a connected component with number of vertices at least $n/4$, then define this connected component as T'. Else, all connected components have at most $n/4$ vertices.

In this case, let T' be the forest formed by the union of a minimum number of connected components of $T \setminus \{u\}$ such that $|T'| \geq n/4$. It is easy to see that $|T'| \leq n/2$. Also, the number of edges in T from $V(T')$ to $V(T) \setminus V(T')$ is at most $\deg_G(u)$, which is at most $c_1 np$.

By property (iii) of Lemma 6.21, the number of edges between $V(T')$ and $V(G) \setminus V(T')$ is $\Omega(n^2 p)$. Each of these edges in G between $V(T')$ and $V(G) \setminus V(T')$ have to contribute to the congestion of at least one of the edges in T between $V(T')$ and $V(G) \setminus V(T')$. Now since T' sends at most $c_1 np$ tree edges to other parts of T, it follows that there exists one edge in T with congestion at least $\Omega(n^2 p)/(c_1 np) = \Omega(n)$, as claimed. □

6.7.1 Constructive Upper Bound for STC of Random Graphs

Let $G = \mathcal{G}(n, p)$ where $p \geq c_0 \log n/n$, and $c_0 = 64$. We prove Theorem 6.13 by giving a polynomial time algorithm to find a spanning tree of G that has $\mathcal{O}(n)$ congestion. The following lemmas show that with high probability G is an (n, s, d_1, d_2, d_3, t)-expanding graph with $s = \Theta(1/p)$, $d_1 = \Theta(1)$, $d_2 = \Theta(np)$, $d_3 = \Theta(np)$, $t = \Theta(1/p)$ (and hence $\delta = o(1)$). The proof of the lemmas are deferred to end of this section.

Lemma 6.24. *For any $S \subseteq V(G)$ such that $|S| = \lceil 1/p \rceil$, we have $|N(S)| \geq c_2 n$ with probability at least $1 - e^{-n/16}$, where $c_2 = 1/25$.*

Lemma 6.25. *For any $S \subseteq V(G)$ such that $|S| \leq 1/p$, we have $|N(S)| \geq c_3 np|S|$ with probability at least $1 - \mathcal{O}(1/n^2)$, where $c_3 = 1/16$.*

Lemma 6.26. *For all $A \subseteq V(G)$ such that $|A| \leq n/2$, and for all $S \subseteq A$, with probability at least $1 - e^{-n}$, S has at least $|S| - c_4/p$ neighbors in $V(G) \setminus A$, where $c_4 = 12$.*

Lemma 6.27. *For all $S \subseteq V(G)$, the cut size $|E(S, V(G) \setminus S)|$ is at most $np|S|$ with probability at least $1 - n^{-c_0/4}$.*

Plugging the bounds from above lemmas into Theorem 6.12, we get Theorem 6.13.
Constants. For easy reference, we list out the constants used in this section.

$$c_0 = 64,\ c_2 = 1/25,\ c_3 = 1/16,\ c_4 = 12$$

Proof of Lemma 6.24: Let $\bar{S} = V(G) \setminus S$. The probability that a fixed vertex in $\bar{S}$ does not have edge to S is at most $(1-p)^{|S|} \leq (1-p)^{1/p} \leq e^{-1}$. Since $|\bar{S}| \geq n - 2/p \geq n - 2n/(c_0 \log n) \geq 31n/32$, the expected value of $|N(S)|$ is at least $(31/32)\, n(1-e^{-1}) \geq n/2$. Hence, using Chernoff bound, the probability that $|N(S)| < c_2 n = n/25$ is at most $e^{-n/8}$. Since the number of such S is at most $n^{2/p} = 2^{2n/c_0} \leq 2^{n/32}$, we have the lemma by applying union bound. □

Proof of Lemma 6.25: Let $\bar{S} = V(G) \setminus S$. Since $|S| \leq 1/p \leq n/\log n$, we have $|\bar{S}| \geq n/2$ for sufficiently large n. Divide $\bar{S}$ into groups of size $\lceil 1/(p|S|) \rceil$. The probability that such a group does not have edge to S is at most $(1-p)^{|S|(1/(p|S|))} \leq 1/e$. The expected number of groups having edge to S is at least $(np|S|/2)(1 - 1/e) \geq np|S|/4$. Thus, by Chernoff bound, the probability that $|N(S)| \leq np|S|/16$ is at most $e^{-np|S|/16} \leq 2^{-c_0|S|\log n/16} \leq 2^{-4|S|\log n}$. The number of sets of size $|S|$ is at most $2^{|S|\log n}$. Hence, taking union bound over all S with $|S| \leq 1/p$, we get the required lemma. □

Proof of Lemma 6.26: First, we prove that for all $C, D \subseteq V(G)$ such that $|C| \geq n/4$,$|D| \geq c_4/p$, and $C \cap D = \emptyset$, there exist at least one edge between C and D with high probability. The probability that there is no edge between such a fixed C and D is at most $(1-p)^{(n/4)(c_4/p)} \leq e^{-c_4 n/4}$. The number of pairs of such C and D is at most 2^{2n}. Hence, by taking union bound, the probability that for all C and D, the claim holds is at least $1 - e^{2n-(c_4 n/4)} \geq 1 - e^{-n}$.

Using the above claim, we prove that for all $S \subseteq A$, S has at least $|S| - c_4/p$ neighbors in $\bar{A} := V(G) \setminus A$ with high probability. Suppose there is an S which violates the claim. Note that we can assume $|S| \geq c_4/p$, because otherwise the claim is vacuously true. Let $B := \bar{A} \setminus N(S)$. There cannot be any edges between S and B. Also, $|B| \geq (n/2) - (|S| - (c_4/p))$. So, $|B|$ is at least c_4/p and when $|B| < n/4$, $|S|$ is at least $n/4$. Hence, using the previous claim, there is an edge between S and B with probability at least $1 - e^{-n}$. Hence, we get a contradiction, and hence our claim is true with probability at least $1 - e^{-n}$. □

Proof of Lemma 6.27: Let $\mathcal{C}(S)$ denote $|E(S, V(G) \setminus S)|$. For a fixed vertex subset S, the expected value of $\mathcal{C}(S)$ is at most $np|S|$. Therefore, probability that $\mathcal{C}(S) > np|S| \geq c_0|S|\log n$ is at most $n^{-c_0|S|/2}$ using Chernoff bounds. The probability that $\mathcal{C}(S) \leq np|S|$ for all sets S of size k is at least $1 - n^{-c_0 k/2 + k} \geq 1 - n^{-c_0/2+1}$ using union bound and using $k \geq 1$. The probability that $\mathcal{C}(S) \leq np|S|$ for all vertex subsets S is at least $1 - n^{-c_0/2+2} \geq$ using union bound over all $k \in [n]$. □

6.8 Open Problems

(1) An important open problem with regard to STC is to find good polynomial time approximation algorithms for STC. Picking any arbitrary spanning tree will give an approximation ratio of n, the number of vertices. There is no polynomial time approximation algorithm known that gives a better approximation ratio, to the best of our knowledge. The best known lower bound for the approximation ratio is $(2 - \varepsilon)$ given by Bodlaender et al. [23].

(2) Does there exist polynomial time algorithms for finding the Győri-Lovász partitions of a k-connected graph? Polynomial time algorithms are only known for $k = 2, 3$ [151, 158]. There are no lower bounds that forbids polynomial time algorithms. It is possible that there is a polynomial time algorithm even for arbitrary k. The problem is a search problem and not a decision problem. We know the required partition exists. Hence the problem is not NP-complete. Our local search algorithm shows that the problem is in complexity class PLS. It is believed that PLS-complete problems do not have polynomial time local search algorithms. It is an interesting question whether finding the Győri-Lovász partitions is a PLS-complete problem. It will be also interesting whether it is PPAD-complete as then it will be unlikely that a polynomial time algorithm exists. PPAD is a class of problems that are guaranteed to have solutions but finding solutions are believed to be hard, for eg., it contains the problem of finding Nash-equilibria.

List of Figures

List of Algorithms

List of Acronyms

ECP	Edge Clique Partition
ECC	Edge Clique Cover
BSD	Binary Symmetric Decompositions
ETH	Exponential Time Hypothesis
FPT	Fixed Parameter Tractable
PTAS	Polynomial Time Approximation Scheme
FPTAS	Fully Polynomial Time Approximation Scheme
LSST	Low Stretch Spanning Tree
STC	Spanning Tree Congestion
FPP	Fitted Partial Partition
SFPP	Semi Fitted Partial Partition
RC	Rainbow Coloring
RVC	Rainbow Vertex Coloring
SRC	Strong Rainbow Coloring
SRVC	Strong Rainbow Vertex Coloring
VSRC	Very Strong Rainbow Coloring
BFS	Breadth First Search
CNF	Conjunctive Normal Form

Bibliography

[1] I. Abraham, Y. Bartal, and O. Neiman. Nearly tight low stretch spanning trees. In *FOCS 2008*, pages 781–790, 2008. 6.1.2

[2] I. Abraham and O. Neiman. Using petal-decompositions to build a low stretch spanning tree. In *STOC 2012*, pages 395–406, 2012. 6.1.2

[3] D. Adolphson and T. C. Hu. Optimal linear ordering. *SIAM Journal on Applied Mathematics*, 25(3):403–423, 1973. 6.1.3

[4] A. Agrawal. Fine-grained complexity of rainbow coloring and its variants. In *Proceedings of the 42nd International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 60:1–60:14, 2017. 5.1.4

[5] M. O. Albertson. You can't paint yourself into a corner. *Journal of Combinatorial Theory, Series B*, 73(2):189 – 194, 1998. 4.4.3

[6] N. Alon, R. M. Karp, D. Peleg, and D. B. West. A graph-theoretic game and its application to the k-server problem. *SIAM J. Comput.*, 24(1):78–100, 1995. (1), 6.1.2

[7] J. Amilhastre, M.-C. Vilarem, and P. Janssen. Complexity of minimum biclique cover and minimum biclique decomposition for bipartite domino-free graphs. *Discrete applied mathematics*, 86(2-3):125–144, 1998. 3.1.1

[8] P. Ananth, M. Nasre, and K. K. Sarpatwar. Rainbow connectivity: Hardness and tractability. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 241–251, 2011. 5.1, 5.1.4

[9] R. Andersen and U. Feige. Interchanging distance and capacity in probabilistic mappings. *CoRR*, abs/0907.3631, 2009. 6.1.3

[10] K. Appel and W. Haken. Every planar map is four colorable. Part I: Discharging. *Illinois J. Math.*, 21(3):429–490, 1977. 4.1

[11] K. Appel, W. Haken, and J. Koch. Every planar map is four colorable. Part II: Reducibility. *Illinois J. Math.*, 21(3):491–567, 1977. 4.1

[12] S. Arora and B. Barak. *Computational complexity: a modern approach.* Cambridge University Press, 2009. 2.5

[13] S. Arora, S. Rao, and U. V. Vazirani. Expander flows, geometric embeddings and graph partitioning. *J. ACM*, 56(2):5:1–5:37, 2009. 6.1.3

[14] F. Ban, V. Bhattiprolu, K. Bringmann, P. Kolev, E. Lee, and D. P. Woodruff. A PTAS for ℓ_p-low rank approximation. *arXiv preprint arXiv:1807.06101*, 2018. 3.1.1, (5)

[15] N. Bansal, U. Feige, R. Krauthgamer, K. Makarychev, V. Nagarajan, J. Naor, and R. Schwartz. Min-max graph partitioning and small set expansion. *SIAM J. Comput.*, 43(2):872–904, 2014. 6.1.3

[16] M. Basavaraju, L. S. Chandran, M. C. Francis, and R. Mathew. Hadwiger's conjecture of total graphs. Private communication. 4.1

[17] M. Basavaraju, L. S. Chandran, D. Rajendraprasad, and A. Ramaswamy. Rainbow connection number and radius. *Graphs and Combinatorics*, 30(2):275–285, 2014. 5.1, 5.1.4

[18] M. Basavaraju, L. S. Chandran, D. Rajendraprasad, and A. Ramaswamy. Rainbow connection number and radius. *Graphs and Combinatorics*, 30(2):275–285, 2014. (4)

[19] D. Bein, L. Morales, W. Bein, C. Shields Jr, Z. Meng, and I. H. Sudborough. Clustering and the biclique partition problem. In *Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS 2008)*, page 475. IEEE, 2008. 1.1.1, 3.1.1

[20] N. Belkale and L. S. Chandran. Hadwiger's conjecture for proper circular arc graphs. *European J. Combin.*, 30(4):946–956, 2009. 1.2, 4.1

[21] S. N. Bhatt, F. R. K. Chung, F. T. Leighton, and A. L. Rosenberg. Optimal simulations of tree machines (preliminary version). In *FOCS 1986*, pages 274–282, 1986. 1.4, 6.1

[22] A. Björklund, T. Husfeldt, and M. Koivisto. Set partitioning via inclusion-exclusion. *SIAM Journal on Computing*, 39:546–563, 2009. 5.6

[23] H. L. Bodlaender, F. V. Fomin, P. A. Golovach, Y. Otachi, and E. J. van Leeuwen. Parameterized complexity of the spanning tree congestion problem. *Algorithmica*, 64(1):85–111, 2012. 6.1.2, (1)

[24] H. L. Bodlaender, K. Kozawa, T. Matsushima, and Y. Otachi. Spanning tree congestion of k-outerplanar graphs. *Discrete Mathematics*, 311(12):1040–1045, 2011. 1.4, 6.1.2

[25] B. Bollobás. *Random Graphs.* Cambridge University Press, 2001. 6.5.1

[26] B. Bollobás, P. Catlin, and P. Erdős. Hadwiger's conjecture is true for almost every graph. *European J. Combin.*, 1(3):195–99, 1980. 1.2, 4.1

[27] B. Bollobás and A. Thomason. Random graphs of small order. *North-Holland Mathematics Studies*, 118:47–97, 1985. 6.7

[28] A. Brandstädt, V. B. Le, and J. P. Spinrad. *Graph Classes: A Survey.* SIAM Monographs in Discrete Mathematics and Applications, 1999. 2.2.1

[29] L. Cai and D. G. Corneil. Tree spanners. *SIAM J. Discrete Math.*, 8(3):359–387, 1995. (3)

[30] Y. Caro, A. Lev, Y. Roditty, Z. Tuza, and R. Yuster. On rainbow connection. *Electron. J. Combin*, 15(1):R57, 2008. 5.1.1, 5.1.4

[31] S. Chakraborty, E. Fischer, A. Matsliah, and R. Yuster. Hardness and algorithms for rainbow connection. *Journal of Combinatorial Optimization*, 21(3):330–347, 2011. 1.3, 5.1, 5.1.4

[32] P. Chalermsook, S. Heydrich, E. Holm, and A. Karrenbauer. Nearly Tight Approximability Results for Minimum Biclique Cover and Partition. In *European Symposium on Algorithms - ESA 2014*, volume 8737 of *LNCS*, pages 235–246. Springer Berlin Heidelberg, 2014. 1.1.2, 3.1.1

[33] L. S. Chandran, Y. K. Cheung, and D. Issac. Spanning Tree Congestion and Computation of Generalized Györi-Lovász Partition. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 32:1–32:14, 2018.

[34] L. S. Chandran, A. Das, D. Issac, and E. J. van Leeuwen. Algorithms and Bounds for Very Strong Rainbow Coloring. In *Latin American Symposium on Theoretical Informatics*, pages 625–639. Springer, 2018.

[35] L. S. Chandran, A. Das, D. Rajendraprasad, and N. M. Varma. Rainbow connection number and connected dominating sets. *Journal of Graph Theory*, 71(2):206–218, 2012. 5.1.1, 5.1.4

[36] L. S. Chandran, D. Issac, and A. Karrenbauer. On the Parameterized Complexity of Biclique Cover and Partition. In *11th International Symposium on Parameterized and Exact Computation (IPEC 2016)*, volume 63 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:13, Dagstuhl, Germany, 2017.

[37] L. S. Chandran, D. Issac, J. Lauri, and E. J. van Leeuwen. Rainbow Coloring and Forest number. Unpublished.

[38] L. S. Chandran, D. Issac, and S. Zhou. Hadwiger's Conjecture for Squares of 2-Trees. *European Journal of Combinatorics*, 76:159 – 174, 2019.

[39] L. S. Chandran, A. Kostochka, and J. K. Raju. Hadwiger number and the cartesian product of graphs. *Graphs and Combinatorics*, 24(4):291–301, 2008. 4.1

[40] L. S. Chandran and D. Rajendraprasad. Rainbow Colouring of Split and Threshold Graphs. In *Proceedings of the 18th Annual International Computing and Combinatorics Conference (COCOON)*, volume 7434 of *Lecture Notes in Computer Science*, pages 181–192. Springer, 2012. 5.1, 5.1.3, 5.1.4

[41] L. S. Chandran and D. Rajendraprasad. Inapproximability of rainbow colouring. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 153–162, 2013. 5.1, 5.1.4, (4)

[42] L. S. Chandran, D. Rajendraprasad, and M. Tesař. Rainbow colouring of split graphs. *Discrete Applied Mathematics*, 216:98–113, 2017. 5.1.3, 5.1.4

[43] G. Chartrand, G. L. Johns, K. A. McKeon, and P. Zhang. Rainbow connection in graphs. *Mathematica Bohemica*, 133(1):85–98, 2008. 1.3, 5.1, 5.1.1, 5.1.2, 5.1.4

[44] G. Chartrand and P. Zhang. *Chromatic graph theory.* CRC press, 2008. 5.1.4

[45] F. Chataigner, L. R. Salgado, and Y. Wakabayashi. Approximation and inapproximability results on balanced connected partitions of graphs. *Discrete Mathematics and Theoretical Computer Science*, 9(1):177–192, 2007. 6.1.3

[46] J. Chen, R. D. Kleinberg, L. Lovász, R. Rajaraman, R. Sundaram, and A. Vetta. (Almost) Tight bounds and existence theorems for single-commodity confluent flows. *J. ACM*, 54(4):16, 2007. 1.4, 1.4.1, 6.1.1, 6.3, 6.4, 6.4.2, 6.4.3, 6.19

[47] L. Chen, X. Li, and H. Lian. Further hardness results on the rainbow vertex-connection number of graphs. *Theoretical Computer Science*, 481:18–23, 2013. 5.1.3

[48] L. Chen, X. Li, and Y. Shi. The complexity of determining the rainbow vertex-connection of a graph. *Theoretical Computer Science*, 412(35):4531–4535, 2011. 5.1.3

[49] M. Chudnovsky and A. O. Fradkin. Hadwiger's conjecture for quasi-line graphs. *J. Graph Theory*, 59(1):17–33, 2008. 1.2, 1.2.1, 1.2.1, 4.1, 4.1

[50] M. Chudnovsky and P. Seymour. Claw-free graphs. VII. Quasi-line graphs. *J. Combin. Theory Ser. B*, 102(6):1267–1294, 2012. 4.1

[51] M. Cygan, F. V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized algorithms*, volume 3. Springer, 2015. 2.5.1

[52] M. Cygan, M. Pilipczuk, and M. Pilipczuk. Known algorithms for edge clique cover are probably optimal. *SIAM Journal on Computing*, 45(1):67–83, 2016. 3.1.1, 3.1.2, 3.6, 3.6, 4, 3.6

[53] P. Dorbec, I. Schiermeyer, E. Sidorowicz, and É. Sopena. Rainbow connection in oriented graphs. *Discrete Applied Mathematics*, 179:69–78, 2014. 5.1

[54] M. E. Dyer and A. M. Frieze. On the complexity of partitioning graphs into connected subgraphs. *Discrete Applied Mathematics*, 10(2):139–153, 1985. 6.1.3

[55] E. Eiben, R. Ganian, and J. Lauri. On the complexity of rainbow coloring problems. In *Proceedings of the 26th International Workshop on Combinatorial Algorithms (IWOCA)*, volume 9538 of *Lecture Notes in Computer Science*, pages 209–220. Springer, 2015. 5.1.2, 5.1.3

[56] E. Eiben, R. Ganian, and J. Lauri. On the complexity of rainbow coloring problems. *Discrete Applied Mathematics*, 246:38–48, 2018. 5.1.3, 5.1.3

[57] J. Ekstein, P. Holub, T. Kaiser, M. Koch, S. Matos Camacho, Z. Ryjáček, and I. Schiermeyer. The rainbow connection number of 2-connected graphs. *Discrete Mathematics*, 313(19):1884–1892, 2013. 5.1.4

[58] M. Elkin, Y. Emek, D. A. Spielman, and S. Teng. Lower-stretch spanning trees. *SIAM J. Comput.*, 38(2):608–628, 2008. 1.4, 6.1.2

[59] A. Ene, W. Horne, N. Milosavljevic, P. Rao, R. Schreiber, and R. E. Tarjan. Fast exact and heuristic methods for role minimization problems. In *SACMAT '08: Proceedings of the 13th ACM symposium on Access control models and technologies*, pages 1–10, New York, NY, USA, 2008. ACM. 1.1.1

[60] D. Eppstein, M. T. Goodrich, and J. Y. Meng. Confluent layered drawings. *Algorithmica*, 47:439–452, 2007. doi:10.1007/s00453-006-0159-8. 1.1.1

[61] P. Erdos, A. W. Goodman, and L. Pósa. The representation of a graph by set intersections. *Canad. J. Math*, 18(106-112):86, 1966. 5.3

[62] P. Erdös, M. Saks, and V. T. Sós. Maximum induced trees in graphs. *Journal of Combinatorial Theory, Series B*, 41(1):61–79, 1986. 5.1.1

[63] U. Feige and R. Krauthgamer. A polylogarithmic approximation of the minimum bisection. *SIAM J. Comput.*, 31(4):1090–1118, 2002. 1.4, 6.1.3

[64] Q. Feng, Z. Zhou, and J. Wang. Parameterized algorithms for maximum edge biclique and related problems. In *International Workshop on Frontiers in Algorithmics*, pages 75–83. Springer, 2016. 3.1.1

[65] P. C. Fishburn and P. L. Hammer. Bipartite dimensions and bipartite degrees of graphs. *Discrete Math.*, 160(1-3):127–148, 1996. 3.1

[66] H. Fleischner, E. Mujuni, D. Paulusma, and S. Szeider. Covering graphs with few complete bipartite subgraphs. *TCS*, 410(21-23):2045 – 2053, 2009. 3.1.1, 3.1.2, 2, 3.2.2, 3.2.2, 3.19

[67] F. V. Fomin, P. A. Golovach, D. Lokshtanov, F. Panolan, and S. Saurabh. Approximation schemes for low-rank binary matrix approximation problems. *arXiv preprint arXiv:1807.07156*, 2018. 3.1.1, (5)

[68] F. V. Fomin, P. A. Golovach, and F. Panolan. Parameterized low-rank binary matrix approximation. *arXiv preprint arXiv:1803.06102*, 2018. 3.1.1

[69] D. S. Franzblau and D. J. Kleitman. An algorithm for covering polygons with rectangles. *Information and control*, 63(3):164–189, 1984. 3.1.1

[70] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1979. 1.1.2, 2.5, 5.4

[71] M. R. Garey, D. S. Johnson, and L. J. Stockmeyer. Some Simplified NP-Complete Graph Problems. *Theoretical Compututer Science*, 1(3):237–267, 1976. 5.1.5

[72] F. Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory, Series B*, 16(1):47 – 56, 1974. 5.3

[73] F. Geerts, B. Goethals, and T. MielikÃďinen. Tiling databases. In *Discovery Science*, pages 278–289. Springer, 2004. 1.1.1

[74] P. A. Golovach, P. Heggernes, P. van 't Hof, and C. Paul. Hadwiger number of graphs with small chordality. *SIAM J. Discrete Math.*, 29(3):1427–1451, 2015. 4.1

[75] M. C. Golumbic. *Algorithmic graph theory and perfect graphs*, volume 57. Elsevier, 2004. 5.3

[76] J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. Data reduction and exact algorithms for clique cover. *Journal of Experimental Algorithmics (JEA)*, 13:2, 2009. 3.1.1

[77] D. A. Gregory, N. J. Pullman, K. F. Jones, and J. R. Lundgren. Biclique coverings of regular bigraphs and minimum semiring ranks of regular matrices. *J. Comb. Theory, Ser. B*, 51(1):73–89, 1991. 3.1, 3.4

[78] H. Gruber and M. Holzer. Inapproximability of Nondeterministic State and Transition Complexity Assuming P$\neq$NP. In *Developments in Language Theory*, volume 4588 of *Lecture Notes in Computer Science*, pages 205–216. Springer, 2007. 1.1.1, 3.1.1

[79] E. Győri. On division of graphs to connected subgraphs. *Colloq. Math. Soc. Janos Bolyai*, 18:485–494, 1976. 1.4, 6.2, 6.1.3, 6.1.4

[80] H. Hadwiger. Über eine Klassifikation der Streckenkomplexe. *Vierteljschr. Naturforsch. Ges. Zürich*, 88:133–142, 1943. 1.2, 4.1, 4.1

[81] M. M. Halldórsson. A still better performance guarantee for approximate graph coloring. *Information Processing Letters*, 45(1):19–23, 1993. (6)

[82] P. Heggernes, D. Issac, J. Lauri, P. T. Lima, and E. J. van Leeuwen. Rainbow Vertex Coloring Bipartite Graphs and Chordal Graphs. In *43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018)*, volume 117 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 83:1–83:13, Dagstuhl, Germany, 2018.

[83] P. Hell and J. Huang. Certifying LexBFS Recognition Algorithms for Proper Interval Graphs and Proper Interval Bigraphs. *SIAM Journal on Discrete Mathematics*, 18(3):554–570, 2004. 5.8

[84] L. Hofer and T. Lambert. Study of the article: An $O(k^2n^2)$ algorithm to find a k-partition in a k-connected graph, 2014. 1

[85] A. Hoyer and R. Thomas. The Győri-Lovász theorem. *arXiv*, abs/1605.01474, 2016. 6.1.3, 6.1.4, 2

[86] Janka Chlebíková. Approximating the maximally balanced connected partition problem in graphs. *Information Processing Letters*, 60(5):225 – 230, 1996. 6.1.3

[87] T. Jiang and B. Ravikumar. Minimal nfa problems are hard. *SIAM Journal on Computing*, 22(6):1117–1141, 1993. 3.1, 3.1.1, (3)

[88] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis. How easy is local search? *J. Comput. Syst. Sci.*, 37(1):79–100, 1988. 6.1.1

[89] S. Jukna and A. S. Kulikov. On covering graphs by complete bipartite subgraphs. *Discrete Mathematics*, 309(10):3399–3403, 2009. 1.1.1

[90] N. Kamčev, M. Krivelevich, and B. Sudakov. Some remarks on rainbow connectivity. *Journal of Graph Theory*, 83(4):372–383, 2016. 5.1.1

[91] F. Kammer and T. Tholey. Approximation algorithms for intersection graphs. *Algorithmica*, 68(2):312–336, 2014. 5.1.5

[92] K. Kawarabayashi and B. Toft. Any 7-chromatic graphs has K_7 or $K_{4,4}$ as a minor. *Combinatorica*, 25(3):327–353, 2005. 4.1

[93] M. Keranen and J. Lauri. Computing minimum rainbow and strong rainbow colorings of block graphs. *arXiv preprint arXiv:1405.6893*, 2014. 5.1.4

[94] M. A. Kiwi, D. A. Spielman, and S. Teng. Min-max-boundary domain decomposition. *Theor. Comput. Sci.*, 261(2):253–266, 2001. 6.1.3

[95] L. T. Kou, L. J. Stockmeyer, and C. K. Wong. Covering edges by cliques with regard to keyword conflicts and intersection graphs. *Commun. ACM*, 21(2):135–139, 1978. 3.1.1

[96] I. Koutis, G. L. Miller, and R. Peng. A nearly $O(m \log n)$ time solver for SDD linear systems. In *FOCS 2011*, pages 590–598, 2011. 6.1.2

[97] Ł. Kowalik, J. Lauri, and A. Socała. On the fine-grained complexity of rainbow coloring. In *Proceedings of the 24th Annual European Symposium on Algorithms (ESA)*, pages 58:1–58:16, 2016. 5.1.4

[98] K. Kozawa and Y. Otachi. Spanning tree congestion of rook's graphs. *Discussiones Mathematicae Graph Theory*, 31(4):753–761, 2011. 1.4, 6.1.2

[99] K. Kozawa, Y. Otachi, and K. Yamazaki. On spanning tree congestion of graphs. *Discrete Mathematics*, 309(13):4215–4224, 2009. 1.4, 6.1.2

[100] M. Krivelevich and R. Yuster. The rainbow connection of a graph is (at most) reciprocal to its minimum degree. *Journal of Graph Theory*, 63(3):185–191, 2010. 1.3, 5.1.1, 5.1.3, 5.1.4, 5.1.5, 5.13, 5.76

[101] E. Kushilevitz and N. Nisan. *Communication Complexity.* Cambridge University Press, New York, NY, USA, 1997. 1.1.1

[102] J. Lauri. *Chasing the Rainbow Connection: Hardness, Algorithms, and Bounds.* PhD thesis, Tampere University of Technology, 2016. 1.3.1, 5.1.1, 5.1.1, 5.1.2, 5.1.2, 5.1.3, 5.1.3, 5.1.3, 5.1.4, 5.1.5, 5.12, 5.3

[103] J. Lauri. Further hardness results on rainbow and strong rainbow connectivity. *Discrete Applied Mathematics*, 201:191–200, 2016. 5.1.4

[104] J. Lauri. Complexity of rainbow vertex connectivity problems for restricted graph classes. *Discrete Applied Mathematics*, 219:132–146, 2017. 5.1.4

[105] H. F. Law, S. L. Leung, and M. I. Ostrovskii. Spanning tree congestions of planar graphs. *Involve*, 7(2):205–226, 2014. 1.4, 6.1.2

[106] D. Li and M. Liu. Hadwiger's conjecture for powers of cycles and their complements. *European J. Combin.*, 28(4):1152–1155, 2007. 1.2, 4.1

[107] H. Li, X. Li, and S. Liu. Rainbow connection of graphs with diameter 2. *Discrete Mathematics*, 312(8):1453–1457, 2012. 5.1.4

[108] S. Li, X. Li, and Y. Shi. Note on the complexity of deciding the rainbow (vertex-) connectedness for bipartite graphs. *Applied Mathematics and Computation*, 258:155–161, 2015. 5.1.3, 5.1.3, 5.1.3, 5.7, 5.72

[109] X. Li, Y. Mao, and Y. Shi. The strong rainbow vertex-connection of graphs. *Utilitas Mathematica*, 93:213–223, 2014. 5.1.3, 5.1.5

[110] X. Li, Y. Shi, and Y. Sun. Rainbow connections of graphs: A survey. *Graphs and Combinatorics*, 29(1):1–38, 2013. 5.1, 5.1.2, 5.1.4

[111] X. Li and Y. Sun. On strong rainbow connection number. *arXiv preprint arXiv:1010.6139*, 2010. 5.1.2

[112] X. Li and Y. Sun. *Rainbow connections of graphs.* Springer Science & Business Media, 2012. 5.1.2

[113] X. Li and Y. Sun. An updated survey on rainbow connections of graphs - a dynamic survey. *Theory and Applications of Graphs*, 0:3, 2017. 1.3, 5.1, 5.1.2, 5.1.4

[114] X. Li and S. Zhou. Labeling outerplanar graphs with maximum degree three. *Discrete Appl. Math.*, 161(1-2):200–211, 2013. 4.6

[115] K.-W. Lih, W.-F. Wang, and X. Zhu. Coloring the square of a K_4-minor free graph. *Discrete Math.*, 269(1âĂŞ3):303–309, 2003. 4.6

[116] B. Lin. The parameterized complexity of k-biclique. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 605–615. SIAM, 2014. 3.1.1

[117] L. Lovász. Coverings and colorings of hypergraphs. In *Proceedings of the 4th Southeastern Conference on Combinatorics, Graph Theory, and Computing*, pages 3–12, 1973. 5.1.5

[118] L. Lovász. A homology theory for spanning trees of a graph. *Acta Math. Acad. Sci. Hungaricae*, 30(3–4):241–251, 1977. 1.4, 6.2, 6.1.3

[119] C. Löwenstein, D. Rautenbach, and F. Regen. On spanning tree congestion. *Discrete Math.*, 309(13):4653–4655, 2009. 1.4, 1.4.1, 6.1.2

[120] A. Lubiw. The boolean basis problem and how to cover some polygons by rectangles. *SIAM Journal on Discrete Mathematics*, 3(1):98–115, 1990. 3.1.1

[121] A. Lubiw. A weighted min-max relation for intervals. *Journal of Combinatorial Theory, Series B*, 53(2):151–172, 1991. 3.1.1

[122] P. Manurangsi. Inapproximability of maximum biclique problems, minimum k-cut and densest at-least-k-subgraph from the small set expansion hypothesis. *Algorithms*, 11(1):10, 2018. 3.1.1

[123] H. Mélot. Facet defining inequalities among graph invariants: The system GraPHedron. *Discrete Applied Mathematics*, 156(10):1875–1891, 2008. 5.1.1

[124] M. Molloy and B. Reed. A bound on the total chromatic number. *Combinatorica*, 18(2):241–280, 1998. 4.1

[125] E. Mujuni and F. Rosamond. Parameterized complexity of the clique partition problem. In *Proceedings of the fourteenth symposium on Computing: the Australasian theory-Volume 77*, pages 75–78. Australian Computer Society, Inc., 2008. 3.1.1, 3.2.3, 3.5, 3.7, (1)

[126] H. Müller. On edge perfectness and classes of bipartite graphs. *Discrete Math.*, 149(1-3):159–187, 1996. 3.1.1

[127] S. Nakano, M. S. Rahman, and T. Nishizeki. A linear-time algorithm for four-partitioning four-connected planar graphs. *Inf. Process. Lett.*, 62(6):315–322, 1997. 1

[128] D. S. Nau, G. Markowsky, M. A. Woodbury, and D. B. Amos. A mathematical analysis of human leukocyte antigen serology. *Math. Biosciences*, 40(3-4):243 – 270, 1978. 1.1.1

[129] I. Nor, D. Hermelin, S. Charlat, J. Engelstadter, M. Reuter, O. Duron, and M.-F. Sagot. Mod/Resc parsimony inference: Theory and application. *Inf. Comput.*, 213:23–32, 2012. 1.1.1, 3.1.1, 3.1.2, 2, 3.1.2

[130] Y. Okamoto, Y. Otachi, R. Uehara, and T. Uno. Hardness results and an exact exponential algorithm for the spanning tree congestion problem. *J. Graph Algorithms Appl.*, 15(6):727–751, 2011. 6.1.2, 6.1.3

[131] J. Orlin. Contentment in graph theory: Covering graphs with cliques. *Indagationes Mathematicae (Proceedings)*, 80(5):406 – 424, 1977. 3.1

[132] M. I. Ostrovskii. Minimal congestion trees. *Discrete Math.*, 285:219–226, 2004. 1.4, 1.4.1, 6.1, (2), 6.1.2, 6.5

[133] M. I. Ostrovskii. Minimum congestion spanning trees in planar graphs. *Discrete Math.*, 310:1204–1209, 2010. 1.4, 6.1.2

[134] M. I. Ostrovskii. Minimum congestion spanning trees in bipartite and random graphs. *Acta Mathematica Scientia*, 31(2):634 – 640, 2011. 1.4.1, 6.1.2, 6.1.2

[135] Y. Pogrow. Solving symmetric diagonally dominant linear systems in sublinear time (and some observations on graph sparsification). Master's thesis, Weizmann Institute of Science, 2017. 6.1.2, 6.1.3

[136] H. Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *STOC*, pages 255–264, 2008. 6.1, 6.1.3

[137] A. Raspaud, O. Sýkora, and I. Vrto. Congestion and dilation, similarities and differences: A survey. In *SIROCCO 2000*, pages 269–280, 2000. 6.1.2

[138] B. Reed and P. Seymour. Hadwiger's conjecture for line graphs. *European J. Combin.*, 25(6):873–876, 2004. 1.2, 1.2.1, 4.1

[139] F. S. Roberts. Applications of edge coverings by cliques. *Discrete applied mathematics*, 10(1):93–109, 1985. 5.1.5

[140] N. Robertson, P. Seymour, and R. Thomas. Hadwiger's conjecture for K_6-free graphs. *Combinatorica*, 13(3):279–361, 1993. 4.1

[141] N. Robertson and P. D. Seymour. Graph minors. II. algorithmic aspects of tree-width. *Journal of Combinatorial Theory, Series B*, 7:309–322, 1986. 5.6

[142] S. E. Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, 2007. 6.1.3

[143] H. U. Simon. On approximate solutions for combinatorial optimization problems. *SIAM J. DISC. MATH.*, 3(2):294 – 310, 1990. 3.1.1

[144] S. Simonson. A variation on the min cut linear arrangement problem. *Mathematical Systems Theory*, 20(4):235–252, 1987. 1.4, 6.1, 6.1.2, 6.1.3

[145] S. Soltan, M. Yannakakis, and G. Zussman. Doubly balanced connected graph partitioning. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1939–1950. Society for Industrial and Applied Mathematics, 2017. 1.4, 6.1.3

[146] D. A. Spielman and S. Teng. A local clustering algorithm for massive graphs and its application to nearly linear time graph partitioning. *SIAM J. Comput.*, 42(1):1–26, 2013. 6.1.3

[147] J. Spinrad, A. Brandstädt, and L. Stewart. Bipartite permutation graphs. *Discrete Applied Mathematics*, 18(3):279–292, 1987. 5.7

[148] D. Steurer. Tight bounds for the min-max boundary decomposition cost of weighted graphs. In *SPAA 2006*, pages 197–206, 2006. 6.1.3

[149] G. Strang. *Introduction to linear algebra*, volume 3. Wellesley-Cambridge Press Wellesley, MA, 1993. 2.3

[150] L. Sunil Chandran, A. Das, D. Rajendraprasad, and N. M. Varma. Rainbow connection number and connected dominating sets. *Journal of Graph Theory*, 71(2):206–218, 2012. 5.1, 5.1.4

[151] H. Suzuki, N. Takahashi, and T. Nishizeki. A linear algorithm for bipartition of biconnected graphs. *Inf. Process. Lett.*, 33(5):227–231, 1990. 6.1.3, (2)

[152] Z. Svitkina and É. Tardos. Min-max multiway cut. In *APPROX-RANDOM 2004*, pages 207–218, 2004. 6.1.3

[153] S. Teng. Scalable algorithms for data and network analysis. *Foundations and Trends in Theoretical Computer Science*, 12(1-2):1–274, 2016. 6.1.3

[154] C. Thomassen. The square of a planar cubic graph is 7-colorable. *J. Combin. Theory, Series B*, 2017. 4.6

[155] K. Uchizawa, T. Aoki, T. Ito, A. Suzuki, and X. Zhou. On the Rainbow Connectivity of Graphs: Complexity and FPT Algorithms. *Algorithmica*, 67(2):161–179, 2013. 5.1.2, 5.1.4

[156] R. Uehara and G. Valiente. Linear structure of bipartite permutation graphs and the longest path problem. *Information Processing Letters*, 103(2):71–77, 2007. 5.7

[157] V. V. Vazirani. *Approximation algorithms.* Springer Science & Business Media, 2013. 2.5.2

[158] K. Wada and K. Kawaguchi. Efficient algorithms for tripartitioning triconnected graphs and 3-edge-connected graphs. In *Graph-Theoretic Concepts in Computer Science, 19th International Workshop, WG '93, Utrecht, The Netherlands, June 16-18, 1993, Proceedings*, pages 132–143, 1993. 6.1.3, (2)

[159] K. Wagner. Über eine Eigenschaft der ebenen Komplexe. *Math. Ann.*, 114(1):570–590, 1937. 4.1, 4.1

[160] G. Wegner. Graphs with given diameter and a coloring problem. *Technical Report, University of Dortmond*, 1977. 4.1, 4.6

[161] D. P. Williamson and D. B. Shmoys. *The design of approximation algorithms.* Cambridge university press, 2011. 2.5.2

[162] D. Wood, G. Xu, and S. Zhou. Hadwiger's conjecture for 3-arc graphs. *Electronic J. Combin.*, 23(4):#P4.21, 2016. 1.2, 4.1

[163] G. Xu and S. Zhou. Hadwiger's conjecture for the complements of Kneser graphs. *J. Graph Theory*, 84(1):5–16, 2017. 1.2, 4.1

[164] D. Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC)*, pages 681–690. ACM, 2006. 5.1.5, 5.4, 5.7