Parameterized Complexity of Graph Partitioning and Geometric Covering

By

Sudeshna Kolay

MATH10201105002

The Institute of Mathematical Sciences, Chennai

A thesis submitted to the

Board of Studies in Physical Sciences

In partial fulfillment of requirements

For the Degree of

DOCTOR OF PHILOSOPHY

of

HOMI BHABHA NATIONAL INSTITUTE

May, 2016

Homi Bhabha National Institute

Recommendations of the Viva Voce Board

As members of the Viva Voce Board, we certify that we have read the dissertation prepared by Sudeshna Kolay entitled "Parameterized Complexity of Graph Partitioning and Geometric Covering" and recommend that it maybe accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy.

_____ Date:

Chair - Prof. Venkatesh Raman

_____ Date:

Guide/Convener - Prof. Saket Saurabh

_____ Date:

Member 1 - Prof. Meena Mahajan

_____ Date:

Member 2 - Prof. V. Arvind

_____ Date:

Member 3 - Prof. Rajiv Raman

Final approval and acceptance of this dissertation is contingent upon the candidate's submission of the final copies of the dissertation to HBNI.

I hereby certify that I have read this dissertation prepared under my direction and recommend that it may be accepted as fulfilling the dissertation requirement.

Date:

Place: Guide

STATEMENT BY AUTHOR

This dissertation has been submitted in partial fulfillment of requirements for an advanced degree at Homi Bhabha National Institute (HBNI) and is deposited in the Library to be made available to borrowers under rules of the HBNI.

Brief quotations from this dissertation are allowable without special permission, provided that accurate acknowledgement of source is made. Requests for permission for extended quotation from or reproduction of this manuscripting whole or in part may be granted by the Competent Authority of HBNI when in his or her judgement the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

Sudeshna Kolay

DECLARATION

I, hereby declare that the investigation presented in the thesis has been carried out by me. The work is original and has not been submitted earlier as a whole or in part for a degree / diploma at this or any other Institution / University.

Sudeshna Kolay

LIST OF PUBLICATIONS ARISING FROM THE THESIS

Journal

1. Esha Ghosh, Sudeshna Kolay, Mrinal Kumar, Pranabendu Misra, Fahad Panolan, Ashutosh Rai, M. S. Ramanujan: Faster Parameterized Algorithms for Deletion to Split Graphs. Algorithmica 71(4):989-1006(2015).

Conferences

1. Sudeshna Kolay, Fahad Panolan, Venkatesh Raman, Saket Saurabh. Parameterized Algorithms on Perfect graphs for deletion to (r, ℓ)-graphs. MFCS 2016: 75:1-75:13.

2. Fedor V. Fomin, Sudeshna Kolay, Daniel Lokshtanov, Fahad Panolan, Saket Saurabh. Subexponential Algorithms for rectilinear Steiner tree and arborescence problems. SOCG 2016: 39:1-39:15.

3. Pradeesha Ashok, Sudeshna Kolay, Saket Saurabh. Parameterized Complexity of Red Blue Set Cover for lines. LATIN 2016:96-109.

4. Sudeshna Kolay, Fahad Panolan. Parameterized Algorithms for Deletion to (r, ℓ)-graphs. FSTTCS 2015:420-433.

5. Pradeesha Ashok, Aditi Dudeja, Sudeshna Kolay. Exact and FPT Algorithms for Max-Conflict Free Coloring in Hypergraphs. ISAAC 2015:271-282.

6. Sudeshna Kolay, Daniel Lokshtanov, Fahad Panolan, Saket Saurabh. Quick but Odd Growth of Cacti. Invited to submit to a special issue of Algorithmica.
The conference version was published in the proceedings of IPEC 2015:258-269.

7. Pradeesha Ashok, Sudeshna Kolay, Neeldhara Misra, Saket Saurabh. Unique Covering Problems with Geometric Sets. COCOON 2015:548-558.

Others

1. Sudeshna Kolay, Fahad Panolan, Saket Saurabh. Communication Complexity of Pairs of Graph Families with Applications. Manuscript.

Sudeshna Kolay

ACKNOWLEDGEMENTS

I would like to thank Professor Saket Saurabh for his guidance of my thesis. I am immensely grateful to him for the numerous insightful discussions, and for the work ethic he tries to instill in his students through example.

I am very grateful to all my coauthors, especially Pradeesha Ashok and Fahad Panolan. It was a wonderful learning experience, working with Pradeesha and Fahad.

I am thankful to all my Professors V. Raman, V. Arvind, Kamal Lodaya, Meena Mahajan, R. Ramanujam, Vikram Sharma, and C. R. Subramaniam, and every other teacher who has taught me, for all that they have taught me.

I would also like to thank my friends and colleagues, without whom my time in IMSc would have been very dull.

Last, but not the least, I would like to thank my parents for the support that they have given me throughout my life.

Contents

List of Figures

List of Algorithms

List of Tables

List of Symbols

$(U, \mathcal{F})$	a set system
(u, v)	the edge between vertices u and v in a multigraph G
$[e_1, \ldots, e_t]$	a path in a multigraph, formed by the sequence of edges $e_1, \ldots, e_t$
$[n]$	the set $\{1, \ldots, n\}$
$[u, \ldots, v]$	a path between vertices u and v in a multigraph G
$[v_1, \ldots, v_t, v_1]$	a cycle in a multigraph G
$\alpha(G)$	size of a maximum independent set in graph G
$\chi(G)$	chromatic number of graph G
log	logarithm function with base 2
$\mathbb{Q}$	the set of rational numbers
$\mathbb{Q}^{\geq 0}$	the set of positive rational numbers
$\mathcal{F}\|_{U'}$	the family obtained when each hyperedge of $\mathcal{F}$ is restricted to elements of $U' \subseteq U$
$\mathcal{G} + ke$	graphs with a set of k edges whose deletion creates a graph in $\mathcal{G}$
$\mathcal{G} + kv$	graphs with a set of k vertices whose deletion creates a graph in $\mathcal{G}$
$\omega(G)$	size of a clique in graph G
$\overline{d}_G(v)$	non-degree of v in G
$\overline{G}$	complement of a graph G
$\overline{K}_n$	stable graph on n vertices
$\overline{N}_G(v)$	non-neighbourhood of v in G
$d_G(v)$	degree of v in G
$E(G)$	edge set of a graph G
$E(v, V_2)$	number of edges between the vertex v and its neighbours in V_2
$E(V_1, V_2)$	number of edges between the vertex subsets V_1 and V_2
f^{-1}	the inverse map of the function f
$G - E'$	subgraph of G where the edge set E' is deleted
$G - V'$	subgraph of G induced by $V(G) \setminus V'$
$G' \leq_m G$	G' is a minor of G
$G' \leq_s G$	G' is a subgraph of G
$G[V']$	subgraph of G, induced by a subset $V' \subseteq V(G)$

$G_1 \cap G_2$	subgraph induced by $V(G_1) \cap V(G_2)$ where G_1 and G_2 are two induced subgraphs of G
$G_1 \cup G_2$	subgraph induced by $V(G_1) \cup V(G_2)$ where G_1 and G_2 are two induced subgraphs of G
$G_1 \uplus G_2$	disjoint union of two graphs G_1 and G_2
$H = (U, \mathcal{F})$	a hypergraph H
K_n	complete graph on n vertices
$N(v)$	neighbourhood of a vertex v when the context of the graph is clear
$N[v]$	closed neighbourhood of v when the context of the graph is clear
$N_G(v)$	neighbourhood of a vertex v in a graph G
$N_G[v]$	closed neighbourhood of v in a graph G
$R(r, \ell)$	Ramsey number for r and ℓ
$U(\mathcal{F}')$	the elements of the universe U present in the family $\mathcal{F}'$
$u \cdots v$	a path between vertices u and v
uPv	the subpath of P that is between vertices u and v
uv	the edge between vertices u and v of a graph G
$V(G)$	vertex set of a graph G
$\mathbb{N}$	the set of non-negative integers $\{1, 2, 3, \ldots\}$
$\mathsf{deg}_H(v)$	the number of hyperedges of H that v belongs to
$\mathsf{Nbd}_H(v)$	the family of hyperedges that contain v

Chapter 1

Introduction

1.1 Preamble

In this thesis, we consider problems in graph partitioning and geometric covering in the realm of Parameterized complexity. Several algorithmic paradigms have been developed in order to cope with the hard problems of classical complexity. However, any algorithmic paradigm meant to cope with the hard problems and to run in polynomial time, such as approximation algorithms or randomized algorithms, must give in to inaccuracy. Parameterized complexity is a fairly new branch of theoretical computer science, with yet another measure of efficiency in terms of running time but where there in no compromise on accuracy. A parameterized problem associates with each input instance, of a classical problem, a parameter, which is usually a positive integer. The aim is to contain the exponential explosion in the running time of algorithms such that the dependence of the exponential function in the running time is only on the parameter associated with the input instance. When carefully chosen, the parameter tends to be much smaller than the input instance. Therefore, we expect parameterized problems to have more efficient algorithms, in terms of running time, than their counterparts in the classical complexity setting.

In this chapter, we first describe the classes and concepts of Parameterized complexity. This is followed by surveys on problems in graph partitioning and geometric covering, studied in Parameterized complexity. The results obtained for graph partitioning problems are described in Part I, while those for geometric covering are described in Part II. In the last section of this chapter, a description of the scope of this thesis and an outline of the organisation of the thesis is given.

1.2 Preliminaries

We begin with a few notations, definitions and concepts that have been used in this thesis.

Notations. We use $[n]$ to denote $\{1, \ldots, n\}$. Similarly, $\mathbb{N} = \{1, 2, \ldots\}$ denotes the set of non-negative integers. The set of rational numbers is denoted by $\mathbb{Q}$, while the set of positive rational numbers is denoted by $\mathbb{Q}^{\geq 0}$. In this thesis, the function log is used to denote the logarithm function with *base* 2. For a function $f : D \to R$ and $y \in R$, we use $f^{-1}(y)$ to denote the set $\{x \in D \mid f(x) = y\}$.

We use standard notations from graph theory [Diestel 2012]. The vertex set and edge set of a graph are denoted as $V(G)$ and $E(G)$ respectively. An edge between two vertices $u, v \in V(G)$ is denoted by uv. Such a pair of vertices are said to be *adjacent* to one another. The complement of the graph G, denoted by $\overline{G}$, has $V(G)$ as its vertex set and $\binom{V(G)}{2} \setminus E(G)$ as its edge set. Here, $\binom{V(G)}{2}$ denotes the family of two sized subsets of $V(G)$. The

neighbourhood of a vertex v, or the set of vertices adjacent to v, is represented as $N_G(v)$, or, when the context of the graph is clear, simply as $N(v)$. The *closed neighbourhood* of a vertex v, denoted by $N[v]$, is the subset $N(v) \cup \{v\}$. The *non-neighbourhood* of a vertex v, or the set of vertices that are not adjacent to v, is denoted by $\overline{N}_G(v)$. The *degree* of a vertex v, or the number of neighbours of v, is denoted by $d_G(v)$. Similarly, the *non-degree* of v, or the number of non-neighbours of v, is denoted by $\overline{d}_G(v)$. An *induced subgraph* of G on the vertex set $V' \subseteq V(G)$ is written as $G[V']$. For a vertex subset $V' \subseteq V(G)$, $G[V(G) \setminus V']$ is also denoted as $G - V'$. Similarly, for an edge set $E' \subseteq E(G)$, $G - E'$ denotes the subgraph G' with $V(G') = V(G)$ and $E(G') = E(G) \setminus E'$. Given two subsets $V_1, V_2 \subseteq V(G)$, $E(V_1, V_2)$ denotes the set of edges in $E(G)$ that have one end point in V_1 and the other in V_2. For a vertex $v \in V(G)$ and subset $V' \subseteq V(G) \setminus \{v\}$, we use $E(v, V')$ to denote the edge set $E(\{v\}, V')$. A path in G, with u and v as endpoints is written as a $u \cdots v$ path. Given a path P, a subpath between vertices $u, v \in V(P)$ is denoted by uPv. A *partition* of G is a tuple $(V_1, V_2, \ldots, V_t)$ of subsets of $V(G)$ such that the disjoint union $V_1 \uplus V_2 \uplus \ldots \uplus V_t = V(G)$. Such a partition is denoted by the tuple $(V_1, V_2 \ldots V_t)$ or by $V_1 \uplus \ldots \uplus V_t$. Each V_i is called a *part*. The subdivision of an edge $e = uv$ of a graph G results in a graph G', with $V(G') = V(G) \cup \{w\}$ and $E(G') = (E(G) \setminus \{e\}) \cup \{uw, wv\}$, where w is a new vertex. A graph $\hat{G}$ is a *subdivision* of a graph G if there is a sequence of graphs $\{G_1, G_2, \ldots, G_t\}$, with $G_1 = G$ and $G_t = \hat{G}$, where for each $1 < i \leq t$, G_i is obtained by the subdivision of an edge of G_{i-1}.

For a graph G, we say a vertex $v \in V(G)$ is a *cut vertex* if $G - \{v\}$ has more connected components than G. A connected graph G' is called a *biconnected* graph if the graph G' does not contain any cut vertex. A *vertex separator* of a connected graph G is a set $S \subseteq V(G)$ such that $G - S$ has at least two connected components. For subsets $A, B \subseteq V(G)$, a minimum (A, B)-vertex separator is the minimum number of vertices to be deleted from G such that A and B belong to different connected components. Similarly, an edge separator of a connected graph G is a set $F \subseteq E(G)$ such that $G - F$ has at least two connected components. For subsets $A, B \subseteq V(G)$, a minimum (A, B)-edge separator is the minimum number of edges to be deleted from G such that A and B belong to different connected components.

We denote by $\omega(G)$ the size of a maximum clique in G. Similarly, $\alpha(G)$ denotes the size of a maximum independent set in G. The chromatic number of G, denoted by $\chi(G)$, is the minimum number of colours required for $V(G)$ such that no two adjacent vertices get the same colour. A subgraph G' of G is denoted as $G' \leq_s G$. Given two induced subgraphs $G_1, G_2 \leq_s G$, $G_1 \cap G_2$ is the induced subgraph $G[V(G_1) \cap V(G_2)]$. Similarly, $G_1 \cup G_2$ denoted the induced subgraph $G[V(G_1) \cup V(G_2)]$. Given two graphs G_1, G_2, $G_1 \uplus G_2$ denotes the graph G, with $V(G) = V(G_1) \cup V(G_2)$ and $E(G) = E(G_1) \cup E(G_2)$. For any positive integers r, ℓ, we use $R(r, \ell)$ to denote the *Ramsey number*. That is, any graph on at least $R(r, \ell)$ vertices has either a clique of size r or an independent set of size ℓ.

We denote the hypergraph as $H = (U, \mathcal{F})$, where U is a universe of n vertices and $\mathcal{F}$ is a family of m subsets of U. We refer to the objects in the universe U by either vertices or elements, and each subset of $\mathcal{F}$ as a *hyperedge*. For any subfamily $\mathcal{F}' \subseteq \mathcal{F}$, we denote the elements present in the subfamily as $U(\mathcal{F}')$. Similarly, for a subset $U' \subseteq U$, $\mathcal{F}|_{U'}$ denotes the family of hyperedges obtained when we restrict each hyperedge of $\mathcal{F}$ to the subset U'. Furthermore, for a vertex $v \in U$, by $\mathsf{deg}_H(v)$ we denote the number of hyperedges the vertex v is part of. The neighbourhood of a vertex $v \in U$, denoted by $\mathsf{Nbd}_H(v)$, is the subfamily of hyperedges in $\mathcal{F}$ that contain v. The tuple $(U, \mathcal{F})$ is also often referred

to as a *set system*. In this thesis as well, we have often used the term set system, when the problem is a direct variant of the Set Cover problem. In other instances, the term hypergraph has been used, when the problem is easier to visualize in the graphic setting.

Tree decompositions and treewidth.

Definition 1.1 (Tree Decomposition [Robertson 1984])**.** *A tree decomposition of a (undirected or directed) graph G, with $V(G)$ as its set of vertices and $E(G)$ as its set of edges, is a tree $\mathbb{T}$ in which each vertex $x \in \mathbb{T}$ has an assigned set of vertices $B_x \subseteq V(G)$ (called a bag) such that $(\mathbb{T}, \{B_x\}_{x\in\mathbb{T}})$ has the following properties:*

- $\bigcup_{x\in\mathbb{T}} B_x = V(G)$
- *For any $uv \in E(G)$, there exists an $x \in \mathbb{T}$ such that $u, v \in B_x$.*
- *If $v \in B_x$ and $v \in B_y$, then $v \in B_z$ for all z on the path from x to y in $\mathbb{T}$.*

In short, we denote $(\mathbb{T}, \{B_x\}_{x\in\mathbb{T}})$ as $\mathbb{T}$.

The *treewidth* $tw(\mathbb{T})$ of a tree decomposition $\mathbb{T}$ is the size of the largest bag of $\mathbb{T}$ minus one. A graph may have several distinct tree decompositions. The treewidth $tw(G)$ of a graph G is defined as the minimum of treewidths over all possible tree decompositions of G.

Graph classes. A complete graph on n vertices is a graph G with edge set $\binom{V(G)}{2}$, and is denoted by K_n. A stable graph on n vertices is a graph G with edge set $\emptyset$, and is denoted by $\overline{K}_n$. An *empty graph* is a graph which does not have any vertices, and therefore no edges as well. A *d-degenerate graph* is an undirected graph in which every induced subgraph has a vertex of degree at most d. A graph G is a *d-regular graph* if for each vertex $v \in V(G)$, $d_G(v) \leq d$. A graph G is a *perfect graph* if, for every induced subgraph H, $\chi(H) = \omega(H)$.

A family $\mathcal{F}$ of graphs is said to be *hereditary*, or *closed under induced subgraphs* if for any graph $G \in \mathcal{F}$, every induced subgraph of G is also contained in $\mathcal{F}$. For a family $\mathcal{G}$ of graphs, $\mathcal{G} + kv$ contains all graphs G such that there is a vertex set $S \subseteq V(G)$, of size at most k, with the property that the graph $G - S \in \mathcal{G}$. Similarly, $\mathcal{G} + ke$ contains all graphs G such that there is an edge set $S \subseteq E(G)$, of size at most k, with the property that the graph $G - S \in \mathcal{G}$.

Algorithm running time bounds. For a given function $g : \mathbb{N} \to \mathbb{R}$, $\mathcal{O}(g(n))$ denotes the set of functions $f : \mathbb{N} \to \mathbb{R}$ for which there exist positive constants c, n_0 such that for all $n \geq n_0$, $0 \leq f(n) \leq cg(n)$. Similarly, $\Omega(g(n))$ denotes the set of functions $f : \mathbb{N} \to \mathbb{R}$ for which there exist positive constants c, n_0 such that for all $n \geq n_0$, $0 \leq cg(n) \leq f(n)$. For more details on asymptotic bounds on functions please refer to [Cormen 2009].

Parameterized Complexity. The goal of parameterized complexity is to find ways of solving NP-hard problems more efficiently than brute force: here the aim is to restrict the combinatorial explosion to a parameter that is hopefully much smaller than the input size. Formally, a *parameterization* of a classical problem is assigning a positive integer parameter k to each input instance. This new bi-variate language is called a *parameterized problem*. We say that a parameterized problem is *fixed-parameter tractable (*FPT*)* if there is an algorithm that solves the problem in time $f(k) \cdot |I|^{\mathcal{O}(1)}$, where $|I|$ is the size of the input and f is an arbitrary computable function depending only on the parameter k. If the

problem has a set Γ of positive integers as parameters, then the problem is called FPT if there is an algorithm solving the problem in $f(\Gamma) \cdot |I|^{\mathcal{O}(1)}$, where $|I|$ is the size of the input and f is an arbitrary computable function depending only on the parameters in Γ. Equivalently, the problem can be considered to be parameterized by $k = \sum_{q \in \Gamma} q$. Such an algorithm is called an FPT algorithm and such a running time is called FPT running time. This class of problems is also called the FPT class.

Parameterized complexity has a well developed sub-field on compression algorithms, called *kernelization*.

Definition 1.2. *A kernelization for a parameterized problem $\Pi \subseteq \Gamma^* \times \mathbb{N}$ is an algorithm which takes $(x, k) \in \Gamma^* \times \mathbb{N}$ as input, runs in time polynomial in $|x| + k$, and outputs a pair $(x', k') \in \Gamma^* \times \mathbb{N}$ such that (a) $(x, k) \in \Pi$ if and only if $(x', k') \in \Pi$ and (b) $|x'|, k' \leq g(k)$, where g is some computable function. The output instance x' is called the kernel, and the function g is referred to as the size of the kernel. If $g(k) = k^{\mathcal{O}(1)}$ $(\mathcal{O}(k))$ then we say that Π has a polynomial (linear) kernel.*

Informally, a kernel for an input instance of Π is an equivalent instance which is small in size. In some cases we may not be able to get a kernelization algorithm. In order to handle this, the concept of a *t-oracle* and *Turing kernelization* was defined [Binkele-Raible 2012].

Definition 1.3. *Given a parameterized problem Π, a t-oracle for Π takes an instance (x, k) of Π and decides in constant time whether it is a* YES *instance.*

Definition 1.4. *For a parameterized problem $\Pi \subseteq \Gamma^* \times \mathbb{N}$ and a computable function g, a* $g(k)$-Turing kernelization *for Π is an algorithm which takes $(x, k) \in \Gamma^* \times \mathbb{N}$ and a $g(k)$-oracle for Π as input, runs in time polynomial in $|x| + k$, and decides whether (x, k) is a* YES *instance of Π. If $g(k) = k^{\mathcal{O}(1)}$ $(\mathcal{O}(k))$ then we say that Π is said to have a polynomial (linear) Turing kernel.*

Sometimes, it might be possible to get an algorithm which outputs polynomially many instances of size bounded by a function of the parameter. These size bounded instances ideally can be solved faster than a large input instance. If at least one of these output instances is equivalent to the input instance, we have a faster algorithm for solving the input instance. Such a kernelization algorithm, which is a special case of a Turing kernelization, is called a *one-many kernelization*.

Definition 1.5. *A one-many kernelization for a parameterized problem $\Pi \subseteq \Gamma^* \times \mathbb{N}$ is an algorithm which takes $(x, k) \in \Gamma^* \times \mathbb{N}$ as input, runs in time polynomial in $|x| + k$, and output pairs $(x_1, k_1), \ldots, (x_r, k_r) \in \Gamma^* \times \mathbb{N}$, where $r \in |x|^{\mathcal{O}(1)}$, such that (a) $(x, k) \in \Pi$ if and only if there exists $i \in [r]$, $(x_i, k_i) \in \Pi$, and (b) For any $i \in [r]$, $|x_i|, k_i \leq g(k)$, where g is some computable function. If $g(k) = k^{\mathcal{O}(1)}$ $(\mathcal{O}(k))$ then we say that Π has a polynomial (linear) one-many kernel.*

It can be shown that a problem is FPT if and only if it has a kernel [Flum 2006]. Depending on the existence of a polynomial kernel for a problem, the FPT class is further refined. Kernelization are also enriched with lower bound theories.

Lower bounds in Kernelization. In recent years, several techniques have been developed to show that certain parameterized problems can not have any polynomial sized kernel unless some classical complexity assumptions are violated. One such technique is the *polynomial parameter transformation*.

Definition 1.6. *Let* Π, Π' *be two parameterized problems. A polynomial time algorithm* $\mathcal{A}$ *is called a polynomial parameter transformation (or ppt) from* Π *to* Π' *if , given an instance* (x, k) *of* Π*,* $\mathcal{A}$ *outputs in polynomial time an instance* (x', k') *of* Π' *such that* $(x, k) \in \Pi$ *if and only if* $(x', k') \in \Pi'$ *and* $k' \leq k^{\mathcal{O}(1)}$*.*

We use the following theorem together with ppt reductions to rule out polynomial kernels.

Proposition 1.1 ([Bodlaender 2011])**.** *Let* Π, Π' *be two parameterized problems such that* Π *is* NP*-hard,* $\Pi \in$ NP *and there exists a polynomial parameter transformation from* Π *to* Π'*. Then, if* Π *does not admit a polynomial kernel neither does* Π'*.*

As an example, CNF-SAT, parameterized by the number of variables, is considered. Here the input is a CNF formula and the problem is to determine whether there is a satisfying assignment for this input formula.

Proposition 1.2 ([Fortnow 2011])**.** CNF-SAT *is* FPT *parameterized by the number of variables; however, it does not admit a polynomial kernel unless* NP $\subseteq$ CoNP*/poly.*

Parameterized Intractability. Similar to the Cook-Levin reductions of classical complexity, there is a notion of parameterized reductions that define a hierarchy on parameterized problems.

Definition 1.7. *Given two parameterized problems* Π, Π'*, a parameterized reduction from* Π *to* Π' *is an* FPT *algorithm that takes an instance* (x, k) *of* Π *and outputs an instance* (x', k') *of* Π' *such that* (x, k) *is a* YES *instance of* Π *if and only if* (x', k') *is a* YES *instance of* Π'*, and* $k' \leq g(k)$ *for a computable function* g*.*

This leads to an accompanying theory of parameterized intractability using which one can identify parameterized problems that are unlikely to admit FPT algorithms.

Similar to classical complexity, parameterized complexity also has a notion of a hierarchy of intractable problems – called the *W hierarchy*. The W hierarchy is a collection of complexity classes. These classes are named $W[1], W[2], \cdots$ and is believed to form a hierarchy FPT$\subset W[1] \subset W[2] \subset \cdots$, where each class is closed under parameterized reductions. For the purpose of this thesis, it is enough to be familiar with a few hard problems in W[1] and W[2]. One can show that a problem is W[1]-hard (W[2]-hard) by presenting a parameterized reduction from a known W[1]-hard problem (W[2]-hard) such as Clique (Set Cover) [Downey 2012] to it.

A parameterized problem is said to be in the class para-NP if it has a nondeterministic algorithm with FPT running time. Notice that if a problem Π is in NP, then any parameterization of Π is a language that belongs to para-NP. It is believed that FPT$\subset W[1] \subset W[2] \subset \cdots \subset$ para-NP. To show that a problem is para-NP-hard, we need to show that the problem is NP-hard when the parameter takes a value from a finite set of positive integers. As an example, 3-Colouring is para-NP-hard parameterized by the number of colours.

For a detailed overview of parameterized complexity, the reader is referred to monographs [Flum 2006, Cygan 2015].

1.3 Graph Partitioning in Parameterized Complexity

Colouring problems have been extensively studied in theoretical computer science. One of the most famous colouring problems is the Planar Colouring problem, where the input graph is a planar graph and the question is to determine the minimum number of colours required for the vertex set such that no two adjacent vertices get the same colour. We can easily determine whether a graph is 2-colourable or not, since it must be a bipartite graph. While we have an upper bound of 4 colours for this problem [Appel 1977], it is NP-hard to determine whether a planar graph is 3-colourable or not [Dailey 1980]. Similarly, the Proper Colouring problem takes a graph G and a positive integer k, and asks whether $V(G)$ can be coloured with k colours such that no adjacent vertices get the same colour. This problem is known to be NP-hard even for $k = 3$ [Garey 2002]. The Subcolouring problem takes as input a graph G and a positive integer k, and asks whether $V(G)$ can be coloured with k colours such that each subset of $V(G)$ that gets the same colour, also known as a *colour class*, induces a disjoint union of cliques. This problem is NP-hard even when $k = 2$ [Broersma 2002]. The Achromatic Number problem takes as input a graph G and a positive integer k, and asks for a colouring of at least k colours such that $V(G)$ has a proper colouring and for any pair of colour classes C_i, C_j, $E(C_i, C_j) \neq \emptyset$. This is known to be NP-hard even on trees [Cairnie 1998]. The b-Chromatic Number problem is related to Achromatic Number. On top of the conditions required for Achromatic Number, this problem requires every colour class to have at least one vertex that has a neighbour in each of the other colour classes. This problem was introduced in [Appel 1977].

In general, a colouring problem can also be thought of as partitioning an input graph while satisfying certain properties. For example, the Proper Colouring problem, with input instance (G, k), is essentially asking whether there is a partitioning of $V(G)$ into k parts, such that each part is an independent set. Some partitioning problems are polynomial time solvable. For example, suppose we want to determine whether an input graph G has a vertex bi-partition (V_1, V_2) such that $G[V_1]$ is a clique and $G[V_2]$ is an independent, then such graphs can be recognised in linear time [Golumbic 2004]. This problem is called Split Graph Recognition and a graph G with such a bipartition is called a *split graph*. Let us focus on a superclass of the class of split graphs. A graph G is an (r, ℓ)-graph if its vertex set can be partitioned into r independent sets and ℓ cliques. Recognising an (r, ℓ)-graph is also known as the Cochromatic Number problem. For $r, \ell \in \{0, 1, 2\}$, recognising an (r, ℓ)-graph requires polynomial time [Feder 2003]. However, when either r or ℓ is at least 3, then the problem becomes NP-hard [Feder 2003]. Another interesting partition, called a Stable Cutset partition of a graph G, is a 3-partition (V_1, V_2, V_3) of $V(G)$ such that V_1 is an independent set and $E(V_2, V_3) = \emptyset$. The problem of recognising the class of graphs that have a stable cutset partition is NP-hard [Klein 1996]. Similarly, a Clique Cutset partition, of a graph G, is a 3-partition (V_1, V_2, V_3) of $V(G)$ such that V_1 is a clique and $E(V_2, V_3) = \emptyset$. However, such a partition can be found in polynomial time [Tarjan 1985]. There are various other vertex partition problems, with conditions not only on the parts but also on pairs of parts, which have been studied in terms of recognition or hardness of recognition [de Figueiredo 2000, MacGillivray 1999, Vikas 2004].

Another variation of a vertex partitioning problem is to describe a homomorphism between the given graph G and a specific graph H. A homomorphism is a function $h : V(G) \to V(H)$ such that for any $uv \in E(G)$, $h(u)h(v) \in E(H)$. Thus, the vertices of $V(H)$ define a partition of $V(G)$. Each part is an independent set of G, and the adjacencies between two parts correspond to the adjacencies between vertices in the graph H. For

example, the PROPER COLOURING problem, on an input instance (G, k), asks whether there is a homomorphism from G to the complete graph K_k. In [Cygan 2016], it was shown that, under standard complexity theoretic assumptions, the best algorithm for finding a homomorphism from an input graph G of n vertices to another graph H is the trivial $2^{\mathcal{O}(n \log n)} n^{\mathcal{O}(1)}$ algorithm of enumerating all possible ordered partitions of $V(G)$ and checking if the partition corresponds to a homomorphism to $V(H)$. This means that not only are the partition problems hard, but there are problems with instances where we do not even expect exact algorithms of the form $c^{|I|}$ for instances I and some fixed constant c.

A natural step to resolving the issue of inefficient algorithms for so many partitioning problems is to study them in the parameterized complexity setting. For example, the COCHROMATIC NUMBER problem is NP-hard even for perfect graphs [Wagner 1984]. In [Heggernes 2013], an FPT algorithm is given for COCHROMATIC NUMBER on perfect graphs. The decision version of ACHROMATIC NUMBER is FPT, parameterized by k, following from [Hell 1976]. On the other hand, b-CHROMATIC NUMBER, parameterized by k, is W[1]-hard [Panolan 2015]. Another direction of research with respect to partitioning problems is to allow a certain number of vertex, or edge, deletions from the input graph such that the resultant graph has the required partition. More formally, let $\mathcal{G}_\Phi$ be the set of all graphs that have at least one partition satisfying a property Φ. Also, suppose there is a polynomial time recognition algorithm for this graph class. Then, it is interesting to ask of the complexity of the recognition problem for the class $\mathcal{G}_\Phi + kv$ or the class $\mathcal{G}_\Phi + ke$.

There is a comprehensive literature on these problems. This genre of problems include some of the most well studied problems in parameterized complexity, such as VERTEX COVER, ODD CYCLE TRANSVERSAL (OCT), EDGE BIPARTIZATION, SPLIT VERTEX DELETION (SVD) and SPLIT EDGE DELETION (SED). VERTEX COVER, in particular, has been extensively studied in the parameterized complexity, and the current fastest algorithm runs in time $1.2738^k n^{\mathcal{O}(1)}$ and has a kernel with $2k$ vertices [Chen 2010]. The parameterized complexity of OCT was a well known open problem for a long time. In 2003, in a breakthrough paper, Reed et al. [Reed 2004] showed that OCT is FPT by developing an algorithm for the problem running in time $\mathcal{O}(3^k mn)$. In fact, this was the first time that the iterative compression technique was used. However, the algorithm for OCT had seen no further improvements in the last 9 years, though several reinterpretations of the algorithm have been published [Hüffner 2009, Lokshtanov 2009]. Only recently, Lokshtanov et al. [Lokshtanov 2014] obtained a faster algorithm for the problem running in time $2.3146^k n^{\mathcal{O}(1)}$ using a branching algorithm based on linear programming. Guo et al. [Guo 2006] designed an algorithm for EDGE BIPARTIZATION running in time $2^k n^{\mathcal{O}(1)}$. Recently, this has been improved to $\mathcal{O}(1.977^k nm)$ [Pilipczuk 2015]. There is another theme of research in parameterized complexity, where the objective is to minimize the dependence of n at the cost of a slow growing function of k. A well known open problem, in the area, was whether OCT admits a linear time parameterized algorithms. Only recently, the first linear time FPT algorithms for OCT on general graphs were obtained, both of which run in time $\mathcal{O}(4^k k^{\mathcal{O}(1)}(m + n))$ [Ramanujan 2014, Iwata 2014]. Kratsch and Wahlström [Kratsch 2014b] obtained a randomized polynomial kernel for OCT and EDGE BIPARTIZATION. Ghosh et al. [Ghosh 2015] studied SVD and SED and designed algorithms with running time $2^k n^{\mathcal{O}(1)}$ and $2^{\mathcal{O}(\sqrt{k} \log k)} n^{\mathcal{O}(1)}$. They also gave the best known polynomial kernels for these problems. Later, Cygan and Pilipczuk [Cygan 2013] designed an algorithm for SVD running in time $1.2738^{k+o(k)} n^{\mathcal{O}(1)}$. Krithika and Narayanaswamy [Krithika 2013] studied VERTEX (r, ℓ)-PARTIZATION prob-

lems on perfect graphs, and among several results they obtain $(r+1)^k n^{\mathcal{O}(1)}$ algorithm for Vertex $(r,0)$-Partization on perfect graphs.

As an example of how tools from parameterized complexity are used to design algorithms, we exhibit the algorithm for SVD from [Ghosh 2015] using the famed iterative compression technique [Reed 2004]

FPT **algorithm for** SVD

A graph G is called a *split graph* if $V(G)$ has a bipartition (V_1, V_2) such that $G[V_1]$ is an induced clique while $G[V_2]$ is an induced independent set. In this case, $(G[V_1], G[V_2])$ is called a *split partition* of G. Any split graph does not contain a 4-cycle (C_4), a 5-cycle (C_5) or the complement of a 4-cycle ($2K_2$) as an induced subgraph. The finite set of graphs $\{C_4, C_5, 2K_2\}$ is said to be a *finite forbidden set* for the class of split graphs. Each graph in the finite forbidden set is referred to as a *forbidden structure.*

We start by stating a lemma, that is implied by Theorem 6.2, [Golumbic 2004].

Lemma 1.1. (Theorem 6.2, [Golumbic 2004]) *A split graph on n vertices can have at most $n+1$ split partitions.*

We will now describe the application of the iterative compression technique to the SVD problem.

Iterative Compression for Split Vertex Deletion**.** Given an instance (G,k) of SVD, we let $V(G) = \{v_1, \ldots, v_n\}$. We define vertex sets $V_i = \{v_1, \ldots, v_i\}$, and let the graph $G_i = G[V_i]$. We iterate through the instances (G_i, k) starting from $i = k+3$. For the i^{th} instance, we try to find a solution $\hat{S}_i$ of size at most k, with the help of a *known* solution S_i of size at most $k+1$. Formally, the compression problem we address is the following.

SVD Compression	**Parameter:** k
Input: Graph G, an SVD set $S \subseteq V$ of size at most $k+1$, integer k	
Question: Does there exist an SVD set of size at most k?	

We reduce the SVD problem to $n-k-2$ instances of the SVD Compression problem as follows. Let $I_i = (G_i, S_i, k)$ be the i^{th} instance. Clearly, the set V_{k+1} is a solution of size at most $k+1$ for the instance I_{k+3}. It is also easy to see that if $\hat{S}_{i-1}$ is a solution of size at most k for instance I_{i-1}, then the set $\hat{S}_{i-1} \cup \{v_i\}$ is a solution of size at most $k+1$ for the instance I_i. We use these two observations to start off the iteration with the instance $(G_{k+3}, S_{k+3} = V_{k+1}, k)$ and search for a solution of size at most k for this instance. If there is such a solution $\hat{S}_{k+3}$, we set $S_{k+4} = \hat{S}_{k+3} \cup \{v_{k+4}\}$ and ask for a solution of size at most k for the instance I_{k+4} and so on. If, during any iteration, the corresponding instance does not have a solution of the required size, it implies that the original instance is also a NO instance. This follows from the fact that if a graph G has a split vertex deletion set of size k, then any vertex induced subgraph of G also has a split vertex deletion set of size k. Finally, the solution for the original input instance will be $\hat{S}_n$. Since there can be at most n iterations, the total time taken to solve the original instance is bounded by n times the time required to solve the SVD Compression problem.

Our algorithm for SVD Compression is as follows. Let the input instance be $I = (G, S, k)$. We guess a subset $Y \subseteq S$ with the intention of picking these vertices in our hypothetical solution for this instance and ignoring the rest of the vertices in S. We delete the set Y from the graph and decrease k appropriately. We then check if the graph $G[S \setminus Y]$ is a split graph and if it is not, then reject this guess of Y as a spurious guess. Suppose that $G[S \setminus Y]$ is indeed a split graph. We now guess and fix a split partition (C_0, I_0) for this graph. By Lemma 1.1, we know that there are at most $k + 2$ such split partitions. The split partition we fix corresponds to the split partition *induced* by the hypothetical solution on the graph $G[S \setminus Y]$. Hence, it now remains to check if there is an SVD set of the appropriate size which is disjoint from $S \setminus Y$, and results in a split graph with a split partition *consistent* with (C_0, I_0). More formally, we have an instance of the following problem.

SVD Compression*	**Parameter:** k
Input: Graph G, an SVD set $S \subset V$ such that $G[S]$ is a split graph, a split partition (C_0, I_0) for the graph $G[S]$, integer k	
Question: Does there exist an SVD set X of size at most k, disjoint from S such that $G - X$ has a split partition consistent with (C_0, I_0)?	

The following lemma gives a polynomial time algorithm for the above problem.

Lemma 1.2. SVD Compression* *can be solved in* $\mathcal{O}(n^3)$ *time.*

Proof. Let S' be a potential solution, and let (C', I') be a fixed split partition for the graph $G - S'$ consistent with the split partition (C_0, I_0). Let (C_1, I_1) be a split partition of the graph $G - S$.

Since we cannot delete edges, at most one vertex of $V(C_1)$ can be contained in $V(I')$ and at most one vertex of $V(I_1)$ can be contained in $V(C')$. Hence, we initially guess these two vertices (either guess could be empty) v_c and v_i where $v_c = V(C_1) \cap V(I')$ and $v_i = V(I_1) \cap V(C')$. We move v_c to I_1 and v_i to C_1. For the sake of convenience we refer to the modified graphs C_1 and I_1 also as C_1 and I_1. Now, let $\hat{I} = I_0 \cup I_1$ and $\hat{C} = C_0 \cup C_1$. It is clear that any vertex in $V(\hat{I})$ which is neighbour to a vertex in $V(I_0) \cup \{v_c\}$ needs to be deleted, and any vertex in $V(\hat{C})$ which is not adjacent to all vertices in $V(C_0) \cup \{v_i\}$ needs to be deleted. Let X be the set of these vertices, that need to be deleted. Now, if X is not disjoint from S, we return NO. On the other hand, we observe that if X is disjoint from S, then deleting X gives us the required kind of split graph. To show that, we consider the partition $((\hat{C} - X), (\hat{I} - X))$ of $G - X$. The graph $\hat{C} - X$ is a clique because $G[V(C_0) \cup \{v_i\}]$ is a clique (otherwise we would have returned NO earlier), $C_1 - X$ is a clique, and all the edges between $V(C_0) \cup \{v_i\}$ and $V(C_1) \setminus X$ are present. Similarly, the graph $\hat{I} - X$ is an independent set because $G[V(I_0) \cup \{v_c\}]$ is an independent set (otherwise we would have returned NO earlier), $I_1 - X$ is an independent set, and no edges between $V(I_0) \cup \{v_c\}$ and $V(I_1) \setminus X$ are present. Hence, if $|X| \leq k$, then we return that it is indeed a YES instance, and return NO otherwise. Guessing the vertices takes $\mathcal{O}(n^2)$ time and in each iteration we spend at most linear time. Hence, the algorithm takes $\mathcal{O}(n^3)$ time. □

Given Lemma 1.2, our algorithm for SVD Compression has a running time of $\mathcal{O}(\Sigma_{i=0}^{k} \binom{k+1}{i} \cdot k \cdot n^{\mathcal{O}(1)}) = \mathcal{O}^*(2^k)$, where the factor of k is due to the number of split partitions of $G[S \setminus Y]$ and $n^{\mathcal{O}(1)}$ is due to the time required to execute our algorithm for SVD Compression*.

Finally, since we solve at most n instances of SVD Compression, our algorithm for SVD runs in time $\mathcal{O}^*(2^k)$, giving us the following theorem.

Theorem 1.1. Split Vertex Deletion *can be solved in time* $\mathcal{O}^*(2^k)$ *time.*

The iterative compression technique has been used several times, in various forms, for designing algorithms in this thesis. We will see more application of this technique in Part I.

1.4 Parameterized Complexity and Computational Geometry

Although computational geometric problems have been studied since long, there has been very little study in the field of parameterized complexity. Very recently, parameterized questions in computational goemetry have started to draw interest. Most likely, one of the first usage of FPT tools in computational geometry was made in [Chambers 2008]. It was shown that the Shortest Splitting Cycle problem on a combinatorial surface, which is NP-hard, is FPT when parameterized by the genus of the surface. However, from the start, parameters have been important in the study of computational geometry. For example, given a set of n points on the 2-dimensional plane $\mathbb{R}^2$, it is possible to find the convex hull of the point set in time $\mathcal{O}(n \log n)$ [Graham 1972]. Despite tight examples for the running time, input-sensitive algorithms for computing the convex hull were sought. In [Kirkpatrick 1986] a $\mathcal{O}(n \log h)$ algorithm was obtain, where h is the number of vertices on the convex hull. Although this is a polynomial time algorithm, h can be thought of as a parameter. Another example is Megiddo's algorithm [Megiddo 1984] for solving a linear programme with d variables and n constraints. The feasible set of such a linear programme can be thought of as a polytope in $\mathbb{R}^d$. Megiddo's algorithm runs in time $\mathcal{O}(2^{2^d}n)$. Later, a $d^{\mathcal{O}(d)}$ algorithm was given in [Dyer , Chazelle 1996]. A randomized algorithm of running time $\mathcal{O}(d^2n + e^{\mathcal{O}(\sqrt{d}\ln d)})$ was also designed in [Matoušek]. In 2016, Chan [Chan 2016] gave a deterministic $d^{(1/2+o(1))d}n$ time algorithm for the problem. All these algorithms are FPT algorithms with d, the number of variables, as the parameter.

The power of parameterized complexity lies in the fact that one is free to choose any aspect of the input as a parameter for the problem. A natural parameter is one which is directly related to the given problem. Otherwise, problems have been studied in parameterized complexity for various non-obvious, or structural parameters [Bodlaender 2014, Jansen 2013, Bodlaender 2013]. In computational geometry, some of the structural parameters that have been used are genus of the embedding surface, combinatorial dimension, distance from triviality, etc. Let us look a little closer to parameterizing by distance from triviality. There are several optimization problems on planar point sets where instances with all points lying on the convex hull can be solved very efficiently. Thus, for such problems, it makes sense to ask how many points are completely inside the convex hull. The number of these points is called the distance from triviality. The Minimum Weight Triangulation problem takes as input a set of points in the plane, and asks for a triangulation with the minimum total length of edges. In [Mulzer 2008], the problem was shown to be NP-hard. When k is the number of points strictly inside the convex hull of the input points, a $(2^k kn^3 + n^3)$ was given in [Spillner 2005]. Recently, a $\mathcal{O}(2^{c\sqrt{k}\log k}k^2n^3)$ algorithm was given in [Knauer 2006]. The problem of Euclidean TSP takes as in-

put n points, and the number k of points strictly inside the convex hull, and ask for a minimum-length network that connects the n given points. In [Deineko 2004], a $\mathcal{O}(k!kn)$ algorithm, using $\mathcal{O}(k)$ space was given. Since these problems are of immense importance in applications, algorithms are designed to optimize the time-space trade-off. The same paper [Deineko 2004] also gives an algorithm running in time $\mathcal{O}(2^k k^2 n)$ and requiring space $2^k kn$.

A persistent bottleneck in many problems on geometric objects on a surface is that, although for surfaces with fixed dimensions the problem has a polynomial time algorithm, the dimension d appears as an exponent in the polynomial. Hence, for surfaces of higher dimensions, the running time is very bad. Many problems such as analysing data sets in areas such as optimization, machine learning, or statistics, have input instances that are embedded in spaces of dimensionality in the order of millions. In such cases, a polynomial time algorithm becomes inefficient. A glimpse of hope lies in finding FPT algorithms parameterized by the dimension of the embedding space. The Red Blue Separation problem takes as input a bipartite universe $U = R \uplus B$, where R is a set of n red points and B is a set of n blue points. The aim is to decide whether there are two hyperplanes such that for any pair of a red and a blue point, the segment between them is intersected by one of the hyperplanes. It was shown in [Giannopoulos 2009] that not only is this problem W[1]-hard when parameterized by d, but under standard complexity theoretic assumptions, this problem cannot have an algorithm running in time $n^{\Omega(\sqrt{(}d))}$. The Ham-sandwich Cuts problem in two dimension takes as input a set of n red points and n blue points in $\mathbb{R}^2$, and asks if there is a line that bisects both sets simultaneously. In two dimension, this problem is linear time solvable [Edelsbrunner 1986]. In [Knauer 2011], it was shown that in higher dimension, the problem is NP-hard, W[1]-hard when parameterized by d, and under standard assumptions requires at least $n^{\Omega(d)}$ running time. Similar results hold for the problem of finding Carotheodory sets, Helly sets among many examples.

Many of the geometric problems are covering problems. For example, given a set of disks one could ask if there are at least k unit disks that do not pair-wise intersect. This can be modelled as a graph, called the intersection graph, where each vertex corresponds to a disk and an edge corresponds to a pair of disks that intersect. Then this problem becomes equivalent to finding an independent set of size at least k in this intersection graph. This problem, parameterized by k, was shown to have a linear kernel in [Alber 2004] when the input disks are such that no disk has too small a radius and no two disks are too close to each other. On the other hand, asking the same parameterized problem for the intersection graphs of unit disks and axis-parallel unit squares is W[1]-hard [Marx 2005]. Similarly, consider the intersection graph of a set of directed segments. Finding an independent set of size k in this intersection graph is FPT parameterized by the number of different directions of the input segments [Marx 2006, Kára 2006]. In fact, in [Kára 2006], the algorithm only works on the intersection graph and does not require a layout of the segments. However, under the standard parameter k, the problem is W[1]-hard [Marx 2006]. The Dominating Set problem has also been shown to be W[1]-hard in the intersection graphs of axis-parallel unit squares, axis-parallel line segments and unit disks on the plane [Marx 2006].

Another type of problems considered in computational geometry are the geometric covering problems. The general aim is to cover a set of n points in a geometric space with the help of at most k specified geometric objects. Many versions of this problem were studied in [Langerman 2005]. The most basic problem was to cover n given points in $\mathbb{R}^2$ with the help of at most k lines. This problem was shown to have a kernel with $\mathcal{O}(k^2)$ points and

therefore to be FPT, when parameterized by k. There are several abstract generalisations given in [Langerman 2005]. A few concrete generalisations are covering points with hyperplanes or with the surface of spheres. These are FPT, parameterized by the number of covering objects. However, when we allow points inside a sphere to be covered by the sphere, then the parameterized problem becomes W[1]-hard [Marx 2005]. Similarly, covering points with at most k unit squares, parameterized by k, is W[1]-hard [Marx 2005]. The dual of this problem is to find a set of at most k points such that a given set of n unit squares are hit, or stabbed, by these points. This problem is also shown to be W[1]-hard, mainly using the duality.

In this thesis, we mainly concern ourselves with covering problems. It is to be noticed that most of the parameterized problems studied till now yield hardness results. Our studies too have a fair share of negative results. This only goes to show the difficulty of the age-old geometric problems and the handicap of the parameters studied thus far.

1.5 Scope of this thesis

This thesis is divided into two main technical parts. The first part comprises results in graph partitioning and the next part comprises results in geometric covering.

Graph Partitioning One graph partitioning problem that is looked at in this thesis is (r, ℓ)-Partitioning. For fixed integers $r, \ell \geq 0$, a graph G is called an (r, ℓ)-*graph* if the vertex set $V(G)$ can be partitioned into r independent sets and ℓ cliques. Such a graph is also said to have *cochromatic number* $r+\ell$. The class of (r, ℓ) graphs generalises r-colourable graphs (when $\ell = 0$). Hence, it is not surprising that recognition of (r, ℓ)-graphs is NP-hard, even when either $r \geq 3$ or $\ell \geq 3$ [Garey 2002].

With r and ℓ as part of the input, the recognition problem is NP-hard even if the input graph is a perfect graph (where the Chromatic Number problem is solvable in polynomial time) [Wagner 1984]. However, it is known to be fixed-parameter tractable (FPT) on perfect graphs when parameterized by r and ℓ. In other words, there is an $f(r+\ell) \cdot n^{\mathcal{O}(1)}$ time recognition algorithm for (r, ℓ)-Partitioning on perfect graphs with n vertices, where f is a function of r and ℓ [Heggernes 2013]. Observe that such an algorithm is unlikely on general graphs as the problem is NP-hard even for constant r and ℓ.

This brings us to the following set of natural parameterized questions: Vertex (r, ℓ)-Partization and Edge (r, ℓ)-Partization. An input to these problems consists of a graph G and a positive integer k and the objective is to decide whether there exists a set $S \subseteq V(G)$ ($S \subseteq E(G)$) such that the deletion of S from G results in an (r, ℓ)-graph. These problems generalise well studied problems such as Odd Cycle Transversal, Edge Odd Cycle Transversal, Split Vertex Deletion and Split Edge Deletion. Chapters 2 and 3 look into Vertex (r, ℓ)-Partization and Edge (r, ℓ)-Partization, parameterized by the number k of allowed deletions.

In fact, given two hereditary graph families $\mathcal{F}_1$ and $\mathcal{F}_2$, a natural question to ask is whether the vertex set of an input graph G can be bipartitioned into (V_1, V_2) such that $G[V_1] \in \mathcal{F}_1$ while $G[V_2] \in \mathcal{F}_2$. Notice that when $\mathcal{F}_1$ is the family of r-colourable graphs and $\mathcal{F}_2$ is the family of all graphs that are complements of ℓ-colourable graphs, then we are asking

for a recognition algorithm for (r, ℓ)-graphs. Similarly, we try to recognise a larger family of graphs, where we can delete at most k vertices to obtain a graph with a required bipartition, with respect to a given pair $(\mathcal{F}_1, \mathcal{F}_2)$. Chapter 4 deals with this generalised version of Vertex (r, ℓ)-Partization.

We deviate a little in Chapter 5. Here, for an input graph G, we are interested in finding a vertex subset that contains at least one vertex from each cycle of even length. This is the Even Cycle Transversal problem. This is not a partitioning problem. In fact, this is a variant of the Hitting Set problem, which is the dual of the Set Cover problem. Therefore, it is closer to a covering problem than a partitioning problem. However, this problem can also be thought of as the complement of the Odd Cycle Transversal problem: while Odd Cycle Transversal asks for a vertex subset S of size at most k, that intersects with (or hits) all odd cycles of the input graph, in Even Cycle Transversal we are looking for a vertex subset S of size at most k that hits all even cycles. Observe that OCT is same as Vertex $(2, 0)$-Partization. While OCT is a very well studied problem, Even Cycle Transversal is lesser known. The relation with OCT motivated the study of Even Cycle Transversal, and hence the deviation.

Geometric Covering The widely studied Set Cover problem takes as input, a universe U of n elements and a family $\mathcal{F}$ of subsets of U. The problem is to determine whether there is a subfamily $\mathcal{F}'$ of at most k subsets such that each element belongs to at least one subset in $\mathcal{F}'$. In other words, the subfamily $\mathcal{F}'$ *covers* all elements of U. Many practical problems can be viewed as the Set Cover problem. In fact, many problems can be viewed as variants of this problem. All the problems of Part II are related to some form of covering.

Almost all variants of Set Cover are hard both in classical complexity as well as parameterized complexity. The motivation of the study conducted in this part is to see if the problems become easier if geometric restrictions are introduced to the set families. In Chapter 6 and 7, we look at direct variants of Set Cover, with the restrictions that the input set systems are those of lines, or hyperplanes, sometimes squares or even sets of bounded intersection.

Chapter 8 looks into Conflict Free Colouring. This is a well-studied vertex colouring problem. The aim is to colour the vertices of an input graph in such a way that for each vertex, there is a neighbour which is uniquely coloured in the neighbourhood. This particular problem arose due to a geometric motivation [Pach 2009]. In this thesis, we look at a variant of the colouring problem, namely the problem of finding a maximum sized subgraph that can be given a conflict-free colouring for a specified number of colours. We show an FPT algorithm for any input graph. This implies algorithms for the special geometric cases.

In Chapter 9, we look at Rectilinear Steiner Tree. The Steiner Tree problem takes as input a graph G and a set of terminals T, and determines whether there is a subgraph of size at most k that covers all the terminals of T. This problem is a special case of the Steiner Tree problem. We give the first subexponential time exact algorithm for this problem. It is worth pointing out that an exact algorithm is also a parameterized algorithm, the number of terminals being the parameter.

Part I

Graph Partitioning

In this part, we describe the results obtained in the problems related to graph partitioning. A graph G is called an (r, ℓ)-graph if the vertex set $V(G)$ can be partitioned into $r+\ell$ parts, r of which are induced cliques and ℓ of which are induced independent sets. Chapter 2 deals with the recognition of graphs which become (r, ℓ)-graphs after deletion of at most k vertices. This problem is called Vertex (r, ℓ)-Partization. We show that this problem has an FPT algorithm, parameterized by k, when $r, \ell \leq 2$. When r or ℓ becomes strictly more than 2, the problem is not expected to have an FPT algorithm, under standard complexity theoretic assumptions. We also address the problem of recognising graphs that become (r, ℓ)-graphs after deletion of at most k edges. We almost complete the dichotomy of this problem, with respect to the constants r, ℓ.

Chapter 3 studies Vertex (r, ℓ)-Partization when the input graphs are restricted to the class of perfect graphs. This is a special case of the problem dealt with in Chapter 2 and we give a FPT algorithm, parameterized by k, r, ℓ.

The results of Chapter 4 lead to a more efficient algorithm of Vertex (r, ℓ)-Partization. However, this is not the focal point of this work. We manage to draw a relation between the two party model of communication complexity and parameterized complexity. As a result of this, we obtain a framework for enumeration algorithms and FPT algorithms for graph partitioning problems of a particular nature, Vertex (r, ℓ)-Partization being one such example.

We deviate a little in Chapter 5. Here, we consider the Even Cycle Transversal problem. Here, for an input graph G, the aim is to determine if there is a set $S \subseteq V(G)$ of size at most k such that $G - S$ does not have any even-length cycle. This is a packing problem, more than a covering or a partitioning problem. On the other hand, it is related to Odd Cycle Transversal, where we have to determine if there is a set $S \subseteq V(G)$ of size at most k such that $G - S$ does not have any odd-length cycles.

Chapter 2

Parameterized Algorithms for Deletion to (r, ℓ)-graphs

2.1 Introduction

As mentioned in Section 1.3, for fixed integers $r, \ell \geq 0$, a graph G is called an (r, ℓ)-*graph* if the vertex set $V(G)$ can be partitioned into r independent sets and ℓ cliques. Although the problem has an abstract setting, some special cases are well known graph classes, and have been widely studied. For example, $(2, 0)$- and $(1, 1)$-graphs correspond to bipartite graphs and split graphs respectively. A $(3, 0)$-graph is a 3-colourable graph. Already, we get a hint of an interesting dichotomy for this graph class, even with respect to recognition algorithms. Throughout the chapter we will use m and n to denote the number of edges and the number of vertices, respectively, in the input graph G. It is well known that we can recognise $(2, 0)$- and $(1, 1)$-graphs in $\mathcal{O}(m + n)$ time. In fact, one can show that recognising whether a graph G is an (r, ℓ)-graph, when $r, \ell \leq 2$, can be done in polynomial time [Brandstdt 1998, Feder 2003]. On the other hand, when either $r \geq 3$ or $\ell \geq 3$, the recognition problem is as hard as the celebrated 3-colouring problem, which is NP-complete [Garey 2002]. These problems are also studied when the input is restricted to be a chordal graph, in which case we can get polynomial time recognition algorithms for every r and ℓ [Feder 2011].

The topic of this chapter is to design recognition algorithms for *almost* (r, ℓ)-graphs in the realm of parameterized algorithms. In particular, we study the following natural parameterized questions on (r, ℓ)-graphs: Vertex (r, ℓ)-Partization and Edge (r, ℓ)-Partization.

Vertex (r, ℓ)-Partization	**Parameter:** k
Input: A graph G and a positive integer k	
Question: Is there a vertex subset $S \subseteq V(G)$ of size at most k such that $G - S$ is an (r, ℓ)-graph?	

Edge (r, ℓ)-Partization	**Parameter:** k
Input: A graph G and a positive integer k	
Question: Is there an edge subset $F \subseteq E(G)$ of size at most k such that $G - F$ is an (r, ℓ)-graph?	

These problems generalise some of the most well studied problems in parameterized complexity, such as Vertex Cover, Odd Cycle Transversal (OCT), Edge Bipartization, Split Vertex Deletion (SVD) and Split Edge Deletion (SED). As we saw in Section 1.3, all these problem have a long history in the fields of algorithm design.

Table 2.1: Summary of known and new results for the family of Vertex (r,ℓ)-Partization problems. New results are highlighted in green (last row).

r,ℓ	Problem Name	FPT	Kernel
$(1,0)$	Vertex Cover	1.2738^k	Poly
$(0,1)$	Vertex Cover on $\overline{G}$	1.2738^k	Poly
$(1,1)$	SVD	$1.2738^{k+o(k)}$	Poly
$(2,0)$	OCT	2.3146^k	Randomized Poly
$(0,2)$	OCT on $\overline{G}$	2.3146^k	Randomized Poly
$(2,1)$, $(1,2)$, $(2,2)$	Vertex $(2,1)$-partization Vertex $(1,2)$-partization Vertex $(2,2)$-partization	3.3146^k	**Randomized Turing Poly**

Table 2.2: Summary of known and new results for the family of Edge (r,ℓ)-Partization problems. New results are highlighted in green.

r,ℓ	Problem Name	FPT	Kernel
$(1,0)$	*Recognisable in polynomial time.*		
$(0,1)$	*Recognisable in polynomial time.*		
$(1,1)$	SED	$2^{\mathcal{O}(\sqrt{k}\log k)}$	Poly
$(2,0)$	Edge Bipartization	2^k	Randomized Poly
$(0,2)$	*Recognisable in polynomial time.*		
$(2,1)$	Edge $(2,1)$-partization	$2^{k+o(k)}$	**Open**
$(1,2)$	Edge $(1,2)$-partization	**FPT**	**Open**
$(2,2)$	Edge $(2,2)$-partization	**Open**	

Our Results. The results in this chapter are taken from [Kolay 2015b]. We do not hope to get FPT algorithms for either Vertex (r,ℓ)-Partization or Edge (r,ℓ)-Partization, when either r or ℓ is at least 3, as the recognition problem itself is NP-complete. This leaves the case of $r,\ell \in \{0,1,2\}$. We almost complete the parameterized complexity dichotomy for these problems by either obtaining new results or using the existing results. We refer to Tables 2.1 and 2.2 for a summary of new and old results. It is worth mentioning that one of our results, namely, $3.3146^k n^{\mathcal{O}(1)}$ time FPT algorithm for Vertex $(2,2)$-Partization (and hence for Vertex $(1,2)$-Partization and Vertex $(2,1)$-Partization) were independently and simultaneously obtained by Baste et al. [Baste 2015].

For both Vertex (r,ℓ)-Partization and Edge (r,ℓ)-Partization, the only new cases for which we need to design new parameterized algorithms to complete the dichotomy is when $(r,\ell) \in \{(1,2),(2,1),(2,2)\}$. Apart from the algorithmic results indicated in the Tables 2.1 and 2.2, we also obtain the following results. When $(r,\ell) \in \{(1,2),(2,1),(2,2)\}$, we obtain an $\mathcal{O}(\sqrt{\log n})$-approximation for these special cases. Finally, we obtain randomized *Turing kernels* for Vertex (r,ℓ)-Partization using this approximation algorithms. In other words, we give a polynomial time algorithm that produces polynomially many instances, $n^{\mathcal{O}(1)}$ of Vertex (r,ℓ)-Partization of size $k^{\mathcal{O}(1)}$ such that with very high probability (G,k) is a YES instance of Vertex (r,ℓ)-Partization if and only if one of the polynomially many instances of Vertex (r,ℓ)-Partization of size $k^{\mathcal{O}(1)}$ is a YES instance. The question of existence of polynomial kernels for these special cases as well as for Edge (r,ℓ)-Partization is open. Even the parameterized complexity of Edge $(2,2)$-Partization remains open.

Our methods. Most of the FPT algorithms are based on the iterative compression technique, and use an algorithm for either OCT or Edge Bipartization as a subroutine. One of the algorithms also uses methods developed in [Marx 2013]. To arrive at the approximation algorithm, we need to take a detour. We start by looking at a slightly larger class of graphs called (r, ℓ)-*split graphs*. A graph G is an (r, ℓ)-split graph if its vertex set can be partitioned into V_1 and V_2 such that the size of a largest clique in $G[V_1]$ is bounded by r and the size of the largest independent set in $G[V_2]$ is bounded by ℓ. Such a bipartition for the graph G is called as (r, ℓ)-split partition. The notion of (r, ℓ)-split graphs was introduced in [Gyárfás 1998]. For any fixed r and ℓ, there is a finite forbidden set $\mathcal{F}_{r,\ell}$ for (r, ℓ)-split graphs [Gyárfás 1998]. That is, a graph G is a (r, ℓ)-split graph if and only if G does not contain any graph $H \in \mathcal{F}_{r,\ell}$ as an induced subgraph. The size of the largest forbidden graph is bounded by $f(r, \ell)$, f being a function given in [Gyárfás 1998]. Since the class (r, ℓ)-graphs is a sub class of (r, ℓ)-split graphs, each graph in $\mathcal{F}_{r,\ell}$ will not appear as an induced subgraph in any (r, ℓ)-graph. For our approximation algorithm, we first make the given graph an (r, ℓ)-split graph by removing the induced subgraphs that are isomorphic to some graph in $\mathcal{F}_{r,\ell}$. Once we have an (r, ℓ)-split graph, we generate an (r, ℓ)-split partition (V_1, V_2) of G. Then we observe that for $r, \ell \in \{1, 2\}$, the problem reduces to finding an approximate solution to Odd Cycle Transversal in $G[V_1]$ and $\overline{G}[V_2]$. Finally, we use the known $\mathcal{O}(\sqrt{\log n})$-approximation algorithm for Odd Cycle Transversal [Agarwal 2005], to obtain a $\mathcal{O}(\sqrt{\log n})$-approximation algorithm for our problems. The Turing kernel for Vertex (r, ℓ)-Partization, when $(r, \ell) \in \{(1, 2), (2, 1), (2, 2)\}$, uses the approximation algorithm and depends on the randomized kernelization algorithm for Odd Cycle Transversal [Kratsch 2014b].

2.2 Preliminaries

We have already seen what (r, ℓ)-graphs are. Below, is a formal definition of the graph class as well as some related definitions.

Definition 2.1. *A graph G is an (r, ℓ)-graph if its vertex set can be partitioned into r independent sets and ℓ cliques. We call such a partition of $V(G)$ an (r, ℓ)-partition. An IC-partition, of an (r, ℓ)-graph G, is a partition (V_1, V_2) of $V(G)$ such that $G[V_1]$ can be partitioned into r independent sets and $G[V_2]$ can be partitioned into ℓ cliques.*

For fixed $r, \ell \geq 0$, the class of (r, ℓ)-graphs is closed under induced subgraphs. The following observation is useful in the understanding of the algorithms presented in the paper. For a graph G, we say $S \subseteq V(G)$ is an (r, ℓ)-*vertex deletion set*, if $G - S$ is an (r, ℓ)-graph.

Observation 2.1. *Let $P = (P_I, P_C)$ and $P' = (P'_I, P'_C)$ be two IC-partitions of an (r, ℓ)-graph G. Then $|P_I \cap P'_C| \leq r\ell$ and $|P'_I \cap P_C| \leq r\ell$.*

Proof. Consider an independent set $I \in P_I$ and a clique $C \in P'_C$. At most 1 vertex of C can also be contained in I. There are at most r independent sets in P_I, so P_I can contain at most r vertices from C. There are at most ℓ cliques in P'_C, each of which can have an intersection of at most r vertices with P_I. Hence, $|P_I \cap P'_C| \leq r\ell$. Similarly, we can prove that $|P'_I \cap P_C| \leq r\ell$. □

2.3 Vertex Deletion to (r, ℓ)-graphs

In this section, we first show that Vertex $(2,2)$-Partization is in FPT, using iterative compression. Then, we explain how to reduce Vertex $(2,1)$-Partization and Vertex $(1,2)$-Partization to Vertex $(2,2)$-Partization. Our algorithm for Vertex $(2,2)$-Partization combines the iterative compression technique with a polynomial bound on the number of IC-partitions of a $(2,2)$-graph. The following Lemma tells about an algorithm to recognise whether a graph is a $(2,2)$-graph, and also about an algorithm to compute all such IC-partitions. These results were shown in several papers [Brandstdt 1998, Feder 2003].

Lemma 2.1. *Given a graph G on n vertices and m edges we can recognise whether G is a $(2,2)$-graph in $\mathcal{O}((n+m)^2)$ time. Also, a $(2,2)$-graph can have at most n^8 IC-partitions and all the IC-partitions can be enumerated in $\mathcal{O}(n^8)$ time.*

For a graph G, we say $S \subseteq V(G)$ is a $(2,2)$-vertex deletion set, if $G - S$ is a $(2,2)$-graph. Now we describe the iterative compression technique and its application to the Vertex $(2,2)$-Partization problem.

Iterative Compression for Vertex $(2,2)$-Partization**.** Let (G,k) be an input instance of Vertex $(2,2)$-Partization and let $V(G) = \{v_1, \ldots, v_n\}$. We define, for every $i \in [n]$, the vertex set $V_i = \{v_1, \ldots, v_i\}$. Denote $G[V_i]$ as G_i. We iterate through the instances (G_i, k) starting from $i = k+5$. Given the i^{th} instances and a known $(2,2)$-vertex deletion set S'_i of size at most $k+1$, our objective is to obtain a $(2,2)$-vertex deletion set S_i of size at most k. The formal definition of this compression problem is as follows.

Vertex $(2,2)$-Partization Compression **Parameter:** k
Input: A graph G and a $k+1$ sized vertex subset $S' \subseteq V(G)$ such that $G - S'$ is a $(2,2)$-graph
Output: A vertex subset $S \subseteq V(G)$ of size at most k such that $G - S$ is a $(2,2)$-graph

We reduce the Vertex $(2,2)$-Partization problem to $n-k-4$ instances of the Vertex $(2,2)$-Partization Compression problem in the following manner. When $i = k+5$, the set V_{k+1} is a $(2,2)$-vertex deletion set of size at most $k+1$ for G_{k+5}. Let $I_i = (G_i, S'_i, k)$ be the i^{th} instance of Vertex $(2,2)$-Partization Compression. If S_{i-1} is a k-sized solution for I_i, then $S_{i-1} \cup \{v_i\}$ is a $(k+1)$-sized $(2,2)$-vertex deletion set for G_i. Hence, we start the iteration with the instance $I_{k+5} = (G_{k+5}, V_{k+1}, k)$ and try to obtain a $(2,2)$-vertex deletion set of size at most k. If such a solution S_{k+5} exists, we set $S'_{k+5} = S_{k+5} \cup \{v_{k+6}\}$ and ask of a k-sized solution for the instance I_{k+6}, and so on. If, during any iteration, the corresponding instance does not have a $(2,2)$-vertex deletion set of size at most k, it implies that the original instance (G,k) is a NO instance for Vertex $(2,2)$-Partization. If the input instance (G,k) is a YES instance, then S_n is a k-sized $(2,2)$-vertex deletion set for G, where $n = |V(G)|$. Since there are at most n iterations, the total time taken by the algorithm to solve Vertex $(2,2)$-Partization is at most n times the time taken to solve Vertex $(2,2)$-Partization Compression. The above explained template for doing iterative compression will be used for approximation algorithms as well as for parameterized algorithms for edge versions of these problems. This template is very similar to that given in Section 1.3, for solving Split Vertex Deletion. Only the base case differs from the previous example.

The following lemma shows that Vertex $(2,2)$-Partization Compression is in FPT. The arguments above imply that Vertex $(2,2)$-Partization is also in FPT.

Lemma 2.2. Vertex $(2,2)$-Partization Compression *can be solved deterministically in time* $3.3146^k|V(G)|^{\mathcal{O}(1)}$.

Proof. We design an algorithm for Vertex $(2,2)$-Partization Compression. Let (G, S') be the instance of the problem and let (P'_I, P'_C) be an IC-partition of $G - S'$. Let S be a *hypothetical solution* of size k for the problem, which the algorithm is supposed to compute. Let (P_I, P_C) be an IC-partition of $G - S$. The algorithm first guesses a partition (Y, N) of S' such that $Y = S' \cap S$ and $N = S' \setminus S$. After this guess, the objective is to compute a set Z of size at most $k' = k - |Y|$ such that $G - (Z \cup Y)$ is a $(2,2)$-graph. Since N is not a part of the solution S, $G[N]$ is a $(2,2)$-graph. Consider the two IC-partitions $(P_I \setminus (S \cup S'), P_C \setminus (S \cup S'))$ and $(P'_I \setminus (S \cup S'), P'_C \setminus (S \cup S'))$ of the $(2,2)$-graph $G - (S \cup S')$. By Observation 2.1, we know that the cardinality of each of the sets $(P_I \cap P'_C) \setminus (S \cup S')$ and $(P_C \cap P'_I) \setminus (S \cup S')$ are bounded by 4. So, the algorithm guesses the sets $V_I = (P_I \cap P'_C) \setminus (S \cup S')$ and $V_C = (P_C \cap P'_I) \setminus (S \cup S')$, each of them having size at most 4. After the guess of V_I and V_C, any vertex in $P'_C \setminus V_I$ either belongs to P_C or belongs to the hypothetical solution S. Similarly, any vertex in $P'_I \setminus V_C$ either belongs to P_I or belongs to the hypothetical solution S. By Lemma 2.1, we know that the number of IC-partitions of $G[N]$ is at most $\mathcal{O}(k^8)$ and these partitions can be enumerated in time $\mathcal{O}(k^8)$. The algorithm now guesses an IC-partition (N_I, N_C) of $G[N]$ such that $N_I \subseteq P_I$ and $N_C \subseteq P_C$. Now consider the partition $(A, B) = ((P'_I \cup N_I \cup V_I) \setminus V_C, (P'_C \cup N_C \cup V_C) \setminus V_I)$ of $V(G) \setminus Y$. Any vertex $v \in A$ either belongs to P_I or belongs to the hypothetical solution S, and any vertex $v \in B$ either belongs to P_C or belongs to the solution S. So the objective is to find two sets $U \subseteq A$ and $W \subseteq B$ such that $G[A \setminus U]$ is a bipartite graph, $G[B \setminus W]$ is the complement of a bipartite graph and $|U| + |W| \leq k'$. As a consequence, the algorithm guesses the sizes k_1 of U and k_2 of W. Then the problem reduces to finding an odd cycle transversal (OCT) of size k_1 for $G[A]$, and an OCT of size k_2 for the complement of the graph $G[B]$. Hence, our algorithm runs the current best algorithm for Odd Cycle Transversal, presented in [Lokshtanov 2014], for finding an OCT U of size k_1 in $G[A]$, and for finding an OCT W of size k_2 in the complement of $G[B]$. This completes the algorithm.

Let $n = |V(G)|$. The algorithm guesses the set $Y = S \cap S'$. First, we fix a set Y of size $k - i$ and compute the running time for the algorithm on this particular guess. The algorithm guesses V_C and V_I, each of size at most 4. The number of such guesses is bounded by $\mathcal{O}(n^8)$. Our algorithm also guesses a partition (N_I, N_C) of $S' \setminus Y$. By Lemma 2.1, the number of such guesses are bounded by k^8. At the end, the algorithm guesses k_1 and k_2 such that $k_1 + k_2 = k - |Y| = i$. Then our algorithm executes algorithm for Odd Cycle Transversal on two instances, running in time $2.3146^{k_1} n^{\mathcal{O}(1)}$ and $2.3146^{k_2} n^{\mathcal{O}(1)}$. Thus the running time, for a particular guess Y, is bounded by $2.3146^i n^{\mathcal{O}(1)}$. The number of guesses for Y, of size i, is exactly $\binom{k+1}{i}$. Since $\sum_{i=0}^{k+1} \binom{k+1}{i} 2.3146^i n^{\mathcal{O}(1)} = 3.3146^k n^{\mathcal{O}(1)}$, the total running time is bounded by $3.3146^k n^{\mathcal{O}(1)}$. □

Lemma 2.2 and the discussions preceding it imply the following theorem.

Theorem 2.1. Vertex $(2,2)$-Partization *can be solved in time* $3.3146^k|V(G)|^{\mathcal{O}(1)}$.

Vertex $(2,1)$-Partization**:** There is a simple reduction from Vertex $(2,1)$-Partization to the Vertex $(2,2)$-Partization problem. Suppose we are given a graph G, where

$|V(G)| = n$. We construct a graph $G' = G \uplus \hat{C}$, where $\hat{C}$ is a clique on $n+3$ new vertices. That is, G' is the disjoint union of G and $\hat{C}$. The next lemma relates the graphs G and G'.

Lemma 2.3. *For any integer $t \leq n$, (G, t) is a* YES *instance of* Vertex $(2,1)$-Partization *if and only if (G', t) is a* YES *instance of* Vertex $(2,2)$-Partization. *Here $G' = G \uplus \hat{C}$ such that $\hat{C}$ is a clique on $n+3$ new vertices that are independent from G.*

Proof. Suppose (G, t) is a YES instance of Vertex $(2,1)$-Partization. Then there is a subset $S \subseteq V(G)$, of size at most t, the deletion of which results in a $(2,1)$-graph G^*. Let G^* have a $(2,1)$-partition $I_1 \cup I_2 \cup C_1$. Then, $I_1 \cup I_2 \cup C_1 \cup \hat{C}$ is a $(2,2)$-partition for $G' - S$. Hence, (G', t) is a YES instance of Vertex $(2,2)$-Partization.

Conversely, suppose (G', t) is a YES instance of Vertex $(2,2)$-Partization. Let $S \subseteq V(G')$ be a $(2,2)$-vertex deletion set of size at most t. The deletion of S from G' results in a $(2,2)$-graph $\tilde{G}$. Let $\tilde{G}$ have a $(2,2)$-partition (I_1, I_2, C_1, C_2), where I_1, I_2 induce independent sets and C_1, C_2 induce cliques in $\tilde{G}$. Since $t \leq n$, and since any independent set I of G' can have at most 1 vertex from $V(\hat{C})$, $|V(\hat{C}) \setminus (S \cup I_1 \cup I_2)| \geq n + 1 - t$. As $\hat{C}$ is disjoint from G, it is only possible that either $C_1 \subseteq V(\hat{C})$ and $C_2 \cap V(\hat{C}) = \emptyset$ or $C_2 \subseteq V(\hat{C})$ and $C_1 \cap V(\hat{C}) = \emptyset$. Without loss of generality, suppose $C_1 \subseteq V(\hat{C})$ and $C_2 \cap V(\hat{C}) = \emptyset$. Then $S' = S \setminus V(\hat{C})$ is of size at most t, and $G - S'$ has a $(2,1)$-partition $(I_1 - V(\hat{C}), I_2 - V(\hat{C}), C_2)$, where $I_1 - V(\hat{C}), I_2 - V(\hat{C})$ induce independent sets and C_2 induces a clique in $G - S'$. Thus, (G, t) is a YES instance of Vertex $(2,1)$-Partization. □

Now if we are given an instance (G, k) of Vertex $(2,1)$-Partization, Lemma 2.3 tells us that it is enough to solve Vertex $(2,2)$-Partization on (G', k). Notice that solving the Vertex $(1,2)$-Partization problem on an input instance (G, k) is equivalent to finding a Vertex $(2,1)$-Partization on $(\overline{G}, k)$, where $\overline{G}$ is the complement graph of G. Thus, we get the following as a corollary of Theorem 2.1.

Corollary 2.1. Vertex $(1,2)$-Partization *and* Vertex $(2,1)$-Partization *have* FPT *algorithms that run in* $3.3146^k n^{\mathcal{O}(1)}$ *time.*

2.4 Approximation algorithm for Vertex (r, ℓ)-Partization

In this section, we give a polynomial time approximation algorithm for the optimization version of Vertex $(2,2)$-Partization, where the aim is to determine the minimum number of vertices that need to be deleted in order to make the input graph G a $(2,2)$-graph. This approximation algorithm has approximation factor $\mathcal{O}(\sqrt{\log n})$. In fact, the main focus of this section is on the design an algorithm for Vertex $(2,2)$-Partization, which takes an instance (G, k), runs in polynomial time and outputs either a solution of size $\mathcal{O}(k^{3/2})$ or concludes that (G, k) is a NO instance. This algorithm is useful for obtaining Turing kernels for Vertex (r, ℓ)-Partization, when $r, \ell \in \{1, 2\}$. Since the reduction from Vertex $(2,1)$-Partization to Vertex $(2,2)$-Partization, in Lemma 2.3, is an approximation preserving reduction, we can get similar approximation algorithms for Vertex $(2,1)$-Partization. Similarly, since Vertex $(1,2)$-Partization on a graph is equivalent

to Vertex $(2,1)$-Partization in the complement graph, we can obtain approximation algorithms for Vertex $(1,2)$-Partization.

We know that (r,ℓ)-graphs is a subclass of (r,ℓ)-split graphs (See Introduction of this Chapter for definition). Now we give a polynomial time algorithm which takes a graph G as input, and outputs an (r,ℓ)-split partition if G is an (r,ℓ)-split graph. We design such an algorithm using iterative compression. Essentially, we show that the following problem, (r,ℓ)-split partition Compression, can be solved in polynomial time.

(r,ℓ)-split partition Compression
Input: A graph G with $V(G) = V \cup \{v\}$ and an (r,ℓ)-split partition (A,B) of $G[V]$
Output: An (r,ℓ)-split partition of G, if G is an (r,ℓ)-split graph, and NO otherwise

Like in the case of the FPT algorithm for Vertex $(2,2)$-Partization, given in Section 2.3, we can show that by running the algorithm for (r,ℓ)-split partition Compression at most $n-2$ times, we can get an algorithm which outputs an (r,ℓ)-split partition of a given (r,ℓ)-split graph. Our algorithm for (r,ℓ)-split partition Compression uses the following simple lemma.

Lemma 2.4. *Let G be an (r,ℓ)-split graph. Let (A,B) and (A',B') be two (r,ℓ)-split partitions of G. Then $|A \cap B'| \leq R(\ell+1,r+1)-1$ and $|A' \cap B| \leq R(\ell+1,r+1)-1$, where the function $R()$ denotes the Ramsey number function.*

Proof. Suppose $|A \cap B'| \geq R(\ell+1,r+1)$. By Ramsey's theorem, we know that $G[A \cap B']$ either contains an independent set of size $\ell+1$ or a clique of size $r+1$. If $G[A \cap B']$ contains an independent set of size $\ell+1$, then $G[B']$ has an independent set of size $\ell+1$. This contradicts our assumption that (A',B') is an (r,ℓ)-split partition of G. Similarly, if $G[A \cap B']$ contains a clique of size $r+1$, then it contradicts our assumption that (A,B) is an (r,ℓ)-split partition of G. This implies that $|A \cap B'| \leq R(\ell+1,r+1)-1$. By similar arguments we can show that $|A' \cap B| \leq R(\ell+1,r+1)-1$. □

Using Lemma 2.4, we show that (r,ℓ)-split partition Compression can be solved in polynomial time for any fixed positive constants r and ℓ.

Lemma 2.5. *For any fixed positive constants r and ℓ, (r,ℓ)-split partition Compression can be solved in polynomial time.*

Proof. Let $(G,(A,B))$ be the given instance of (r,ℓ)-split partition Compression, where $V(G) = V \cup \{v\}$ and (A,B) is a (r,ℓ)-split partition of $G[V]$. Let $n = |V(G)|$. Let (A',B') be a hypothetical solution for the problem. Since $G[V]$ is a subgraph of G, $(A' \setminus \{v\}, B' \setminus \{v\})$ is an (r,ℓ)-split partition of $G[V]$. Thus, by Lemma 2.4, we know that $|A \cap B'| \leq R(\ell+1,r+1)-1$ and $|A' \cap B| \leq R(\ell+1,r+1)-1$. So our algorithm guesses the sets $U = A \cap B'$ and $W = A' \cap B$, each of size at most $R(\ell+1,r+1)$. The total number of possible choices for U and W is clearly bounded by $n^{2R(\ell+1,r+1)}$. For the correct guess of U and W, $A' \setminus \{v\} = (A \cup W) \setminus U$ and $B' \setminus \{v\} = (B \cup U) \setminus W$. Let $X = (A \cup W) \setminus U$ and $Y = (B \cup U) \setminus W$. Thus, it is enough to check whether one of $(X \cup \{v\}, Y)$ or $(X, Y \cup \{v\})$ is a valid (r,ℓ)-split partition of the graph G, and output the result. This can be tested in time $n^{r+\ell}$ time. Since there are $n^{2R(\ell+1,r+1)}$ choices for the guess U and W, the total running time is bounded by $\mathcal{O}(n^{2R(\ell+1,r+1)+r+\ell})$. This completes the proof of the lemma. □

By applying Lemma 2.5, at most $n-2$ times, we can get the following lemma.

Lemma 2.6. *For any fixed constants r and ℓ, there is an algorithm which takes a graph G as input, runs in polynomial time, and decides whether G is an (r,ℓ)-split graph. Furthermore, if G is an (r,ℓ)-split graph then the algorithm outputs an (r,ℓ)-split partition (V_1,V_2) of G.*

We know that any (r,ℓ)-graph is also an (r,ℓ)-split graph. The following lemma gives a relation between an (r,ℓ)-split partition and an IC-partition of a (r,ℓ)-graph.

Lemma 2.7. *Let G be an (r,ℓ)-graph. Let (A,B) be an IC-partition of G and (A',B') be an (r,ℓ)-split partition of G. Then $|A\cap B'|\leq r\ell$ and $|A'\cap B|\leq r\ell$.*

Proof. Suppose $|A\cap B'|\geq r\ell+1$. Since (A,B) is an IC-partition of an (r,ℓ)-graph G, we know that A can be partitioned into r independent sets. Also, since $|A\cap B'|\geq r\ell+1$, by Pigeonhole Principle, there is an independent set I in A such that $|I\cap B'|\geq \ell+1$. This implies that the size of the largest independent set in B' is at least $\ell+1$, contradicting our assumption that (A',B') is an (r,ℓ)-split partition of G. Hence we have shown that $|A\cap B'|\leq r\ell$. By similar arguments we can show that $|A'\cap B|\leq r\ell$. □

Before giving an approximation algorithm for Vertex (r,ℓ)-Partization, we need to mention about a polynomial time approximation algorithm for Odd Cycle Transversal and a finite forbidden characterization of (r,ℓ)-graphs. Using the FPT algorithm for OCT [Kratsch 2014b], and an $\mathcal{O}(\sqrt{\log n})$-approximation algorithm for OCT [Agarwal 2005], one can prove the following proposition.

Proposition 2.1 ([Kratsch 2014b])**.** *There is a polynomial time algorithm which takes a graph G and an integer k as input and outputs either an OCT, of G, of size at most $\mathcal{O}(k^{3/2})$ or concludes that there is no OCT of size k for G.*

For any fixed r and ℓ, there is a finite forbidden set $\mathcal{F}_{r,\ell}$ for (r,ℓ)-split graphs [Gyárfás 1998]. That is, a graph G is an (r,ℓ)-split graph if and only if G does not contain any graph $H\in\mathcal{F}_{r,\ell}$ as an induced subgraph. The size of the largest forbidden graph is bounded by $f(r,\ell)$, f being a function given in [Gyárfás 1998]. Since $f(2,2)$ is a constant, it is possible to compute the forbidden set $\mathcal{F}_{r,\ell}$ in polynomial time: the forbidden graphs are of size at most $f(2,2)$. Since the class of (r,ℓ)-graphs is a sub class of (r,ℓ)-split graphs, each graph in $\mathcal{F}_{r,\ell}$ will not appear as an induced subgraph in any (r,ℓ)-graph. Now we are ready to design a polynomial time approximation algorithm for Vertex $(2,2)$-Partization.

Theorem 2.2. *There is an algorithm which takes a graph G and an integer k as input, runs in polynomial time and outputs either a set S of size $\mathcal{O}(k^{3/2})$ such that $G-S$ is a $(2,2)$-graph, or concludes that (G,k) is a* NO *instance of* Vertex $(2,2)$-Partization.

Proof. The algorithm first finds a maximal set $\mathcal{T}$ of vertex disjoint subgraphs of G, such that each subgraph in $\mathcal{T}$ is isomorphic to a graph in $\mathcal{F}_{2,2}$. If $|\mathcal{T}|>k$, then clearly (G,k) is a NO instance of Vertex $(2,2)$-Partization. So the algorithm will output NO if $|\mathcal{T}|>k$. Now consider the graph $G'=G-V(\mathcal{T})$. Here, $V(\mathcal{T})$ denotes the set of vertices appearing in graphs in $\mathcal{T}$. Since $\mathcal{T}$ is a maximal set of vertex disjoint subgraphs in G which are isomorphic to graphs in $\mathcal{F}_{2,2}$, G' must be a $(2,2)$-split graph.

Next, our algorithm finds a set $S \subseteq V(G')$ of size bounded by $\mathcal{O}(k^{3/2})$, such that $G' - S$ is a $(2,2)$-graph. Since G' is a subgraph of G, if (G,k) is a YES instance of VERTEX $(2,2)$-PARTIZATION, then (G',k) is also a YES instance. Let S^* be an hypothetical solution of the instance (G',k) of VERTEX $(2,2)$-PARTIZATION, and let (A,B) be an IC-partition of $G' - S^*$. Now our algorithm applies Lemma 2.6 on graph G' and computes a $(2,2)$-split partition (A',B') of G' in polynomial time. By Lemma 2.7, we know that $|A \cap B'| \leq 4$ and $|A' \cap B| \leq 4$. So the algorithm will guess the set $U = A \cap B'$ and $W = A' \cap B$. The number of possible guesses for U and W is bounded by n^8. For the correct guess of U and W, we know that $A = (A' \cup U) \setminus (W \cup S^*)$ and $B = (B' \cup W) \setminus (U \cup S^*)$. Consider the partition (V_1, V_2) of $V(G')$, where $V_1 = (A' \cup U) \setminus W$ and $V_2 = (B' \cup W) \setminus U$. For the correct guess of U and W, we know that each vertex in V_1 either belongs to A or belongs to S^* and each vertex in V_2 either belongs to B or belongs to S^*. Now to compute a solution for (G',k), it is enough to find an OCT S_1 in $G[V_1]$, and an OCT S_2 in the complement graph of $G'[V_2]$, such that $|S_1| + |S_2| = k$. Our algorithm applies Proposition 2.1 on $G'[V_1]$, and on the complement graph of $G'[V_2]$. If these algorithms output an OCT S_1 and an OCT S_2 for graphs $G'[V_1]$ and $\overline{G'}[V_2]$, then $S_1 \cup S_2$ is of size bounded by $\mathcal{O}(k^{3/2})$ and $G' - (S_1 \cup S_2)$ is a $(2,2)$-graph. Since $G' = G - V(\mathcal{T})$ and $G' - (S_1 \cup S_2)$ is a $(2,2)$-graph, we know that $G - (S_1 \cup S_2 \cup V(\mathcal{T}))$ is a $(2,2)$-graph. So, our algorithm outputs $S_1 \cup S_2 \cup V(\mathcal{T})$ as the required output. Since $|V(\mathcal{T})| \leq k \cdot f(2,2)$, it is true that $|S_1 \cup S_2 \cup V(\mathcal{T})| = \mathcal{O}(k^{3/2})$. If the algorithm mentioned in Proposition 2.1 returns NO for all possible guesses of U and W, then our algorithm outputs NO. It is easy to see that the number of steps in our algorithm is bounded by a polynomial in $|V(G)|$. □

Using the arguments of Theorem 2.2, we can also design an approximation algorithm for finding a minimum $(2,2)$-vertex deletion set of a graph G. Let S be an optimum $(2,2)$-vertex deletion set, and (A,B) be the corresponding IC-partition of $G' = G - S$. Let $\mathcal{T}$ be a maximal set of vertex disjoint subgraphs of G, that are each isomorphic to a graph in $\mathcal{F}_{2,2}$. The number of subgraphs in $\mathcal{T}$ is at most $|S|$ and the number of vertices involved in these forbidden subgraphs is at most $f(2,2)|S|$. The remaining graph G' is a $(2,2)$-split graph. Using Lemma 2.6, we can find a $(2,2)$-split partition (A',B') of G'. Let $(\hat{A},\hat{B})$ be the restriction of (A,B) to G'. As argued above, at most 4 vertices from A' could be part of $\hat{B}$. Let this set of 4 vertices be called U. The rest of the vertices of A' either belong to $\hat{A}$ or to S. The set $U \cup (S \cap A')$ is an OCT for A', of size at most $4 + |S \cap A'|$. The algorithm of [Agarwal 2005] returns an $\mathcal{O}(\sqrt{\log n})$-approximate ODD CYCLE TRANSVERSAL solution S_1 for $G[A']$, which has to be of size at most $(4 + |S \cap A'|) \cdot \mathcal{O}(\sqrt{\log n})$. There is a similar property on the vertices of B'. Applying the algorithm of [Agarwal 2005], on $\overline{G'[B']}$, returns an $\mathcal{O}(\sqrt{\log n})$-approximate ODD CYCLE TRANSVERSAL solution S_2, which has to be of size at most $(4+|S \cap B'|) \cdot \mathcal{O}(\sqrt{\log n})$. Thus, $V(\mathcal{T}) \cup S_1 \cup S_2$ is a $(2,2)$-vertex deletion set of G, with size at most $(f(2,2) + \mathcal{O}(\sqrt{\log n}))|S|$. This, together with Lemma 2.3 and the discussion after that, lead to the following theorem.

Theorem 2.3. VERTEX $(2,1)$-PARTIZATION, VERTEX $(1,2)$-PARTIZATION, *and* VERTEX $(2,2)$-PARTIZATION *admit polynomial time approximation algorithms with factor* $\mathcal{O}(\sqrt{\log n})$.

2.5 Turing Kernels for Vertex Deletion to (r, ℓ)-graphs

In this section, we give a randomized Turing kernel for the problem Vertex $(2,2)$-Partization. The equivalence in Lemma 2.3 ensures that there is a randomized Turing kernel for Vertex $(2,1)$-Partization. Since, Vertex $(1,2)$-Partization on (G,k) is equivalent to Vertex $(2,1)$-Partization on $(\overline{G},k)$, a randomized Turing kernel for Vertex $(1,2)$-Partization follows.

We have seen in Section 2.3 that eventually, the algorithm for Vertex $(2,2)$-Partization runs two instances of OCT. In this section, we explain that we can use the kernelization of OCT to get a one-many kernel for Vertex $(2,2)$-Partization. This also gives us a Turing kernel for Vertex $(2,2)$-Partization. A randomized polynomial kernel for OCT was shown by Kratsch and Wahlström [Kratsch 2012], using the concept of representative family. They showed that it is possible to find a set of $k^{\mathcal{O}(1)}$ "relevant" vertices, of the input graph, which contains an optimum solution. This leads to a randomized kernel for OCT. In fact, the following lemma follows from the work of Kratsch and Wahlström.

Lemma 2.8. *Let G be a graph and X be an OCT of G. There is a randomized polynomial time algorithm which computes a set $Z \subseteq V(G)$ of size $\mathcal{O}(|X|^3)$ such that for any $Y \subseteq X$, a minimum sized OCT of $G - Y$ is fully contained in Z.*

Proof. We give an outline of the proof of this Lemma. The results in [Reed 2004] show that the Odd Cycle Transversal problem is equivalent to many instances of the problem of find a minimum vertex separator in an auxiliary graph of the input graph. This is the main idea used to show that Odd Cycle Transversal is in FPT. Let G be a graph and X is an OCT of G. Note that $G - X$ is a bipartite graph. Without loss of generality we may assume that X is independent, otherwise we can subdivide any edge within X and still maintain X as an OCT of G. Let $S_1 \uplus S_2$ be a bipartition of $G - X$. The auxiliary graph G' of G, constructed in [Reed 2004], is as follows. We define $V(G')$ to be $(V(G) \setminus X) \cup \{x_1, x_2 | x \in X\}$. For a set $U \subseteq X$, $X'(U) = \{x_1, x_2 | x \in U\}$. The edge set $E(G')$ is defined as follows: $E(G - X)$ is contained in $E(G')$. Additionally, we add edges between x_1 and neighbours of x in S_2, and between x_2 and neighbours of x in S_1.

Given $U \subseteq X$, a valid partition of $X'(U)$ is pair (S,T) which satisfies the following properties.

1. $S \uplus T = X'(U)$;
2. For every $x \in U$, $|\{x_1, x_2\} \cap S| = |\{x_1, x_2\} \cap T| = 1$;

Note that for a subset $U \subseteq X$ and a valid partition (S,T) of $X'(U)$, $\delta(G' - X'(X \setminus U), S, T)$ denotes the size of a minimum (S,T)-separator in $G' - X'(X \setminus U)$. The following lemma is derived from the results of Reed et al. [Reed 2004].

Lemma 2.9. *Let G be a graph and X be an OCT of G. Let $Y \subseteq X$ and (S,T) be a valid partition of $X'(Y)$. Let W' be a minimum (S,T)-separator in $G' - X'(X \setminus Y)$. It is also true that $|X \setminus Y| + |W'| = \min\{|X \setminus U| + \delta(G' - X'(X \setminus U), S', T')\}$, where $U \subseteq X$, and (S', T') is a valid partition of $X'(U)$. Then, the set $(X \setminus Y) \cup (W' \setminus X) \cup \{x | x \in X, \{x_1, x_2\} \cap W' \neq \emptyset\}$ is a minimum OCT for G.*

The following *cut covering lemma* is proved in [Kratsch 2012].

Lemma 2.10. *Let G be a graph, and $X \subseteq V(G)$ be a set of terminals. We can identify, in randomized polynomial time, a set Z of $O(|X|^3)$ vertices such that for any $S, T, R \subseteq X$, a minimum (S,T)-vertex separator in $G - R$ is contained in Z.*

Now the proof of the Lemma follows directly from Lemma 2.9 and Lemma 2.10. □

Now, we explain our Turing kernel for Vertex $(2,2)$-Partization using Lemma 2.8. Given an instance (G,k) of Vertex $(2,2)$-Partization, first we construct $|V(G)|^{\mathcal{O}(1)}$ many instances of a problem which is in NP, and each instance has size bounded by a polynomial in k. Then, by using the Cook-Levin theorem [Cook 1971], we can reduce each of these instances to instances of Vertex $(2,2)$-Partization, and thus arrive at a one-many kernelization for Vertex $(2,2)$-Partization. This also implies that we have a Turing kernel for the problem. We first run the polynomial time approximation algorithm described in Theorem 2.2. If the approximation algorithm outputs NO, then the algorithm will output a trivial NO instance of the problem. Otherwise, let X be the solution returned by the approximation algorithm on input (G,k). We know that the cardinality of X is bounded by $\mathcal{O}(k^{3/2})$. Now we fix an IC-partition (P_I, P_C) of $G - X$. Let S be a *hypothetical* solution of size at most k and (Q_I, Q_C) be an IC-partition of $G - S$. It follows from Observation 2.1 that $|P_I \cap Q_C| \leq 4$ and $|Q_I \cap P_C| \leq 4$. This observation leads to the following lemma.

Lemma 2.11. *(G,k) is a* YES *instance of* Vertex $(2,2)$-Partization *if and only if there exist $V_C \subseteq P_I$ and $V_I \subseteq P_C$, each of cardinality at most 4, such that $X' = X \cup V_C \cup V_I$ can be partitioned into (X'_I, X'_D, X'_C), with the following properties:*

1. *There is a set $Z_I \subseteq (P_I \setminus V_C) \cup X'_I$ such that $Z_I \cup X'_D \cup X'_C$ is an OCT for $G[P_I \cup X']$. In other words, Z_I is an OCT for $G[P_I \cup X'_I]$.*
2. *There is a set $Z_C \subseteq (P_C \setminus V_I) \cup X'_C$ such that $Z_C \cup X'_D \cup X'_I$ is an OCT for $\overline{G}[P_C \cup X']$. In other words, Z_C is an OCT for $\overline{G}[P_C \cup X'_C]$.*
3. *$|Z_I \cup Z_C \cup X'_D| \leq k$.*

Proof. Suppose (G,k) is a YES instance of Vertex $(2,2)$-Partization. Then there is a k-sized solution Z such that $G - Z$ is a $(2,2)$-graph. Let (Q_I, Q_C) be an IC-partition of $G - Z$. Let $V_C = P_I \cap Q_C$ and $V_I = P_C \cap Q_I$. It follows from Observation 2.1 that $|V_I| \leq 4$ and $|V_C| \leq 4$. Notice that any vertex in $P_I \setminus V_C$ either belongs to Q_I or to Z. Similarly, any vertex in $P_C \setminus V_I$ either belongs to Q_C or to Z. Let $X' = X \cup V_I \cup V_C$. Now we define $X'_I = X' \cap Q_I$, $X'_C = X' \cap Q_C$ and $X'_D = X' \cap Z$. Let $Z_I = Z \cap P_I$ and $Z_C = Z \cap P_C$. Note that $Z_I \cap V_C = \emptyset$ and $Z_C \cap V_I = \emptyset$. From the definition of X', V_I and V_C, it is clear that $V_I \subseteq X'_I$ and $V_C \subseteq X'_C$. Since $V_C \subseteq X'_I$ and $V_C \subseteq X'_C$, it is true that $(P_I \cup X') \setminus (Z_I \cup X'_D \cup X'_C) = Q_I$. Also since, $G[Q_I]$ is a bipartite graph, it must be the case that $(Z_I \cup X'_D \cup X'_C)$ is an OCT of $G[P_I \cup X']$. By similar arguments, we can show that $(Z_C \cup X'_D \cup X'_I)$ is an OCT of $\overline{G}[P_C \cup X']$. Since $Z_I \cup Z_C \cup X'_D = Z$ and $|Z| = k$, the set $Z_I \cup Z_C \cup X'_D$ satisfies condition 3 in the lemma. This completes the proof of the forward direction.

Conversely, suppose there are sets $V_C \subseteq P_I$ and $V_I \subseteq P_C$, each of size at most 4, such that $X' = X \cup V_I \cup V_C$ has a 3-partition (X'_I, X'_D, X'_C) with the properties mentioned in the

lemma. That is, there is an OCT Z_I for the graph $G[P_I \cup X'_I]$ and an OCT Z_C for the graph $\overline{G}[P_C \cup X'_C]$ such that $|Z_I \cup Z_C \cup X'_D| \leq k$. Then we claim that $Z = Z_I \cup Z_C \cup X'_D$ is a $(2,2)$-vertex deletion set of G. Consider the sets $Q_I = (P_I \cup X'_I) \setminus Z_I$ and $Q_C = (P_C \cup X'_C) \setminus Z_C$. By our assumption $G[Q_I]$ and $\overline{G}[Q_C]$ are bipartite graphs. Also note that $Q_I \uplus Q_C \uplus Z = V(G)$. Hence Z is a $(2,2)$-vertex deletion set of G and (Q_I, Q_C) is an IC-partition of $G - Z$. □

The Lemma 2.11 allows us to reduce an instance of Vertex $(2,2)$-Partization to polynomially many instances of a problem which is in NP. Consider the following problem.

Twin Odd Cycle Transversal (TOCT) **Parameter:** k
Input: Two graphs G_1 and G_2, terminals $X \subseteq V(G_1)$, $Y \subseteq V(G_2)$, a bijection Φ between X and Y, and an integer k
Question: Is there a partition of X into three parts (X_1, X_D, X_2) such that there is an OCT $Z_1 \subseteq V(G_1) \setminus (X_D \cup X_2)$ for the graph $G_1 - (X_D \cup X_2)$, an OCT $Z_2 \subseteq V(G_2) \setminus (\Phi(X_D) \cup \Phi(X_1))$ for the graph $G_2 - (\Phi(X_D) \cup \Phi(X_1))$, and $\lvert Z_1 \cup X_D \cup Z_2 \rvert \leq k$?

Clearly the problem TOCT is in NP. Because of Lemma 2.11, for each $V_C \subseteq P_I$ and $V_I \subseteq P_C$ of cardinality at most 4, we construct an instance of TOCT, of size bounded by a polynomial in k, using Lemma 2.8. For a fixed pair $V_C \subseteq P_I$ and $V_I \subseteq P_C$, each of cardinality at most 4, we describe the corresponding instance of TOCT. Let $X' = X \cup V_I \cup V_C$. Recall that X is the approximate solution of size bounded by $\mathcal{O}(k^{3/2})$, and (P_I, P_C) is the IC-partition of $G - X$ that we fixed. Note that X' is a $(2,2)$-vertex deletion set of G and $(P_I \setminus V_C, P_C \setminus V_I)$ is an IC-partition of $G - X'$. The following observation is derived from the fact that $(P_I \setminus V_C, P_C \setminus V_I)$ is an IC-partition of $G - X'$ and $V_I \cup V_C \subseteq X'$.

Observation 2.2. *The set X' is an OCT of $G[P_I \cup X']$ and also an OCT of $\overline{G}[P_C \cup X']$.*

For a particular choice of $V_C \subseteq P_I$ and $V_I \subseteq P_C$ of cardinality at most 4, we construct an instance of TOCT as follows. Let $X' = X \cup V_I \cup V_C$. Let $G_1 = G[P_I \cup X']$ and $G_2 = \overline{G}[P_C \cup X']$. By Observation 2.2, X' is an OCT in graphs G_1 and G_2. We apply Lemma 2.8 and get a set of relevant vertices $Z_1 \subseteq V(G_1)$ of size bounded by $\mathcal{O}(k^{9/2})$. Next, we construct a graph G_1^* as follows: delete all the vertices $V(G_1) \setminus (X' \cup Z_1)$ from G_1. Add two length (three length) path between two vertices in $V(G_1^*)$, if there is an even length (odd length) path between the corresponding vertices in G_1 using only vertices from $V(G) \setminus (X' \cup Z_1)$. Similarly, we construct a graph G_2^* from G_2. We output $H = (G_1, G_2, X', X', k)$ as the reduced instance of TOCT, with the bijection between X' and X' being the natural identity map. Since there are $\mathcal{O}(n^4)$ choices for selecting V_C and V_I, our algorithm will output instances $H_1, H_2, \ldots H_t$, where $t = \mathcal{O}(n^4)$ and the size of each H_i is bounded by $\mathcal{O}(k^9)$.

Using Lemmata 2.8 and 2.11, we can prove that the above one-many reduction is correct.

Lemma 2.12. *(G, k) is a* YES *instance of* Vertex $(2,2)$-Partization *if and only if there exists i such that H_i is a* YES *instance of* TOCT.

Proof. Let (G, k) be a YES instance. Recall that X is an approximate solution and (P_I, P_C) is an IC-partition of $G - X$.

By Lemma 2.11, we know that there exists $V_C \subseteq P_I$ and $V_I \subseteq P_C$ such that the set $X' = X \cup V_I \cup V_C$ can be partitioned into (X'_I, X'_D, X'_C) with the following properties.

- there is set $Z_I \subseteq P_I \setminus V_C$ such that Z_I is an OCT of $G[P_I \cup X'_I]$.
- there is set $Z_C \subseteq P_C \setminus V_I$ such that Z_C is an OCT of $\overline{G}[P_C \cup X'_C]$.
- $|Z_I \cup Z_C \cup X'_D| \leq k$

In our reduction, we have constructed an instance H_i corresponding to the sets V_C and V_I. That is, H_i is constructed from the graphs $G_1 = G[P_I \cup X']$ and $G_2 = G[P_C \cup X']$. In the construction of H_i, we first constructed G_1^* from G_1 and G_2^* from $\overline{G_2}$, by finding relevant vertices Y_1 and Y_2 in graph G_1 and G_2 respectively, using Lemma 2.8. Finally we consider the graph $H_i = (G_1^*, G_2^*, X', X', k)$.

From the construction of G_1^* and using Lemma 2.8, we know that $G_1^* - (X'_D \cup X'_C)$ has an OCT Z_I^* of size at most $|Z_I|$, because Z_I is an OCT in $G_1 - (X'_D \cup X'_C)$. Similarly $G_2^* - (X'_D \cup X'_I)$ has an OCT Z_C^* of size at most $|Z_C|$, because Z_C is an OCT in $G_2 - (X'_D \cup X'_I)$. This implies that $|Z_I^* \cup Z_C^* \cup X'_D| \leq k$. Thus, H_i is a YES instance of TOCT.

In the converse direction, suppose there is an i such that the instance H_i is a YES instance of TOCT. Note that H_i is constructed for a particular $V_C \subseteq P_I$ and $V_I \subseteq P_C$, each of cardinality at most 4. Let $X' = X \cup V_C \cup V_I$. That is the instance $H_i = (G_1^*, G_2^*, X', X', k)$ where G_1^* is constructed from $G_1 = G[X' \cup P_I]$ and G_2^* is constructed from $G_2 = \overline{G}[P_C \cup X']$. By our assumption H_i is a YES instance of TOCT. This implies that there is a partition of X' into (X'_1, X'_D, X'_2), and that there exists an OCT Z_I^* of $G_1^* - (X'_D \cup X'_2)$ and an OCT Z_C^* of $G_2 * -(X'_D \cup X'_1)$ such that $|Z_I^* \cup Z_C^* \cup X'_D| \leq k$. By Lemma 2.8, there exists an OCT Z_I of $G_1 - (X'_D \cup X'_2)$ of size at most $|Z_I^*|$ and an OCT Z_C of $G_2 - (X'_D \cup X'_1)$ of size at most $|Z_C^*|$. Thus, $|Z_I \cup Z_C \cup X'_D| \leq k$ and the conditions in the Lemma 2.11 are met by the partition of X' in to (X'_1, X'_D, X'_2). This implies that (G, k) is a YES instance of Vertex $(2,2)$-Partization. □

The problem TOCT is in NP and Vertex $(2,2)$-Partization is NP-complete. Therefore, by Cook-Levin theorem each instance H_i of TOCT can be reduced to an an instance of Vertex $(2,2)$-Partization in polynomial time. Also, the size of each instance H_i is bounded by $\mathcal{O}(k^{9/2})$. Thus, we obtain the following theorem.

Theorem 2.4. *There is a randomized polynomial Turing kernel for* Vertex $(2,2)$-Partization.

Since there is a parameter preserving reduction from Vertex $(2,1)$-Partization and Vertex $(1,2)$-Partization to Vertex $(2,2)$-Partization, the following corollary is derived from Theorem 2.4.

Corollary 2.2. *There is a randomized polynomial Turing kernel for* Vertex $(2,1)$-Partization *and* Vertex $(1,2)$-Partization.

2.6 Edge Deletion to (r, ℓ)-graphs

In this section, we show that Edge $(2,1)$-Partization and Edge $(1,2)$-Partization are in FPT.

2.6.1 Edge $(2,1)$-Partization

In this subsection, we show that Edge $(2,1)$-Partization is in FPT, using iterative compression. The iteration step is again similar to that in Section 2.3. For Edge $(2,1)$-Partization, the corresponding compression problem is defined as follows.

Edge $(2,1)$-Partization Compression **Parameter:** k
Input: A graph G with $V(G) = V \cup \{v\}$, an integer k and an edge set $S' \subseteq E(G-\{v\})$, of size at most k, such that $G[V] - S'$ is a $(2,1)$-graph
Output: A subset $S \subseteq E$ of size at most k such that $G - S$ is a $(2,1)$-graph

As in the case of Vertex $(2,2)$-Partization, we can show that Edge $(2,1)$-Partization can be solved, by running Edge $(2,1)$-Partization Compression at most $|V(G)|$ times, for an input instance (G,k). The following lemma is useful for our purpose.

Lemma 2.13. *Let G be a graph on n vertices, $v \in V(G)$ and $|E(G-\{v\})| \leq k$. Then the number of cliques in G is bounded by $2^{\mathcal{O}(\sqrt{k})}n$ and these cliques can be enumerated in time $2^{\mathcal{O}(\sqrt{k})}n$.*

Proof. First, we bound the size of a maximum clique in G by $\mathcal{O}(\sqrt{k})$. Let ℓ be the size of a maximum clique in $G-\{v\}$. Since the number of edges in $G-\{v\}$ is at most k, we know that $\binom{\ell}{2} \leq k$. This implies that ℓ is bounded above by $\sqrt{8k}-1$. Since the size of a largest clique in G is at most one more than the largest clique in $G-\{v\}$, the size of a maximum clique in G is bounded by $\sqrt{8k}$. It is well known that a graph H on k edges is $\sqrt{2k}$-degenerate. This implies that G is $\sqrt{2k}+1$ degenerate (Lemma 5.10, [Cygan 2015]). Now, we know from [Eppstein 2010] that G has at most $n3^{\sqrt{2k}+1}$ maximal cliques in G and can be enumerated in time $\mathcal{O}(\sqrt{k}n3^{\sqrt{2k}})$. Since every clique in G has size at most $\sqrt{8k}$, given a maximal clique C of G, we can generate all the cliques contained in C (by enumerating all subsets of C) in time proportional to $2^{\mathcal{O}(\sqrt{k})}$. This implies that the number of cliques in G is upper bounded by $2^{\mathcal{O}(\sqrt{k})}n$ and it can be enumerated in time $2^{\mathcal{O}(\sqrt{k})}n$. □

Next we show that Edge $(2,1)$-Partization Compression is in FPT.

Lemma 2.14. Edge $(2,1)$-Partization Compression *can be solved in time $2^{k+o(k)}n^{\mathcal{O}(1)}$.*

Proof. Let (G,k,S') be the input instance and $|V(G)| = n$. If $G - S'$ is a $(2,1)$-graph, then we return S'. Otherwise we do the following. Let S be a *hypothetical solution* for the problem and let (P_I, P_C) be an IC-partition of $G - S$, which the algorithm is supposed to compute. Let $G' = G[V] - S'$. Since G' is a $(2,1)$-graph, the vertex set V can be partitioned to I_1, I_2 and C such that $G'[I_1]$ and $G'[I_2]$ are graphs with no edges, and $G'[C]$ is a complete graph.

Since $G' = G[V] - S'$, and $I_1 \subseteq V$ and $I_2 \subseteq V$ induce independent sets in G', we have $E(G[I_1]) \subseteq S'$ and $E(G[I_2]) \subseteq S'$. Also, since $|S'| \leq k$, we have $|E(G[I_1])| \leq k$ and $|E(G[I_2])| \leq k$. Consider the partition of the vertex set of G, $V \cup \{v\}$, into three parts $(I_1 \cup \{v\}, I_2, C)$. Recall that (P_I, P_C) is an IC-partition of our hypothetical solution S. Our algorithm guesses the sets of vertices $A = (I_1 \cup \{v\}) \cap P_C$ and $B = I_2 \cap P_C$. Since the partition P_C should induce a clique, $A \cup B$ should also induce a clique. Thus, guessing the vertex sets A and B from $I_1 \cup \{v\}$ and I_2 respectively is equal to guessing two cliques

from $G[I_1 \cup \{v\}]$ and $G[I_2]$ such that together they form a clique in G. By Lemma 2.13, the number of cliques in $G[I_1 \cup \{v\}]$ and $G[I_2]$ is bounded by $2^{\mathcal{O}(\sqrt{k})}n$ and these cliques can be enumerated in time $2^{\mathcal{O}(\sqrt{k})}n$.

After guessing A and B, we know that in our hypothetical IC-partition (P_I, P_C), $A \cup B \subseteq P_C$ and $(I_1 \cup I_2 \cup \{v\}) \setminus (A \cup B) \subseteq P_I$. Let $C' = \{u \in C \mid A \cup B \subseteq N(u)\}$. The following claim implies that we can set $P_C = A \cup B \cup C'$.

Claim 2.1. *If there is a subset $S_1 \subseteq E(G)$ and a partition (P'_I, P'_C) of $V(G)$ such that (i) $G - S_1$ is a $(2,1)$-graph, (ii) (P'_I, P'_C) is an IC-partition of $G - S_1$, (iii) $(I_1 \cup I_2 \cup \{v\}) \setminus (A \cup B) \subseteq P'_I$ and (iv) $A \cup B \subseteq P'_C$, then there is a subset $S_2 \subseteq E(G)$ and a partition (P''_I, P''_C) of $V(G)$ such that (a) $|S_2| \leq |S_1|$, (b) $G - S_2$ is a $(2,1)$-graph, (c) (P''_I, P''_C) is an IC-partition of $G - S_2$, (d) $(I_1 \cup I_2 \cup \{v\}) \setminus (A \cup B) \subseteq P''_I$ and (e) $A \cup B \cup C' = P''_C$.*

Proof. Suppose we are given a set S_1 and an IC-partition (P'_I, P'_C) of $G - S_1$ with the properties mentioned in the statement of the claim. Since (P'_I, P'_C) is an IC-partition of $G - S_1$, S_1 is an Edge Odd Cycle Transversal set of $G[P'_I]$. Since $(I_1 \cup I_2 \cup \{v\}) \setminus (A \cup B) \subseteq P'_I$, we know that $P'_C \setminus (A \cup B) \subseteq C$. Also since P'_C is a clique and $A \cup B \subseteq P'_C$, we know that $P'_C - (A \cup B) \subseteq C'$. Consider the partition (P''_I, P''_C) of $V(G)$, where $P''_C = A \cup B \cup C'$ and $P''_I = V(G) \setminus P''_C$. Note that P''_C is a clique and $P''_I \subseteq P'_I$. This implies that the edge set $S_2 = S_1 \cap E(G[P''_I])$ is an Edge Odd Cycle Transversal set of $G[P''_I]$. Hence the set S_2 and the partition (P''_I, P''_C) are the required set and the partition, respectively, in the claim. □

Claim 2.1 implies that for the correct guess of A and B, we can set $P_C = A \cup B \cup C'$. This in turn implies that the problem is now reduced to that of deleting the minimum number of edges to make the graph $G-(A \cup B \cup C')$ bipartite.This is the Edge Odd Cycle Transversal problem on $(G-(A \cup B \cup C'), k)$. The problem Edge Odd Cycle Transversal can be solved in time $2^k n^{\mathcal{O}(1)}$, where n is the number of vertices in the input graph [Guo 2006]. Since there are $2^{\mathcal{O}(\sqrt{k})}n^2$ choices for guessing A and B, the total running time of the algorithm is bounded by $2^{k+o(k)}n^{\mathcal{O}(1)}$. □

Thus, by using Lemma 2.14, we prove the following theorem.

Theorem 2.5. Edge $(2,1)$-Partization *can be solved in time* $2^{k+o(k)}n^{\mathcal{O}(1)}$.

2.6.2 Edge $(1,2)$-Partization

In this subsection, we show that Edge $(1,2)$-Partization is in FPT. Again we use the iterative compression technique to solve the problem. For our algorithm, we need an algorithm for the following version of Odd Cycle Transversal. Let $\mathcal{G}$ be an hereditary graph class such that $\mathcal{G}$ is decidable. Then the problem $\mathcal{G}$-Weighted Bipartition is defined as follows.

$\mathcal{G}$-Weighted Bipartition	**Parameter:** $k + W$
Input: A graph G, $w : V(G) \to \mathbb{N}^+$ and integers k and W	
Output: An OCT O of G, of size at most k such that $w(O) \leq W$ and $G[O] \in \mathcal{G}$	

Here, the weight function w can be extended over subsets of $V(G)$ in a natural way: for a subset $V' \subseteq V(G)$, $w(V') = \Sigma_{v \in V'} w(v)$. Marx et al. [Marx 2013] showed that the unweighted version of the problem, named, $\mathcal{G}$-Bipartition can be solved in FPT time. The proof by Marx et al., constructs an "equivalent graph" with treewidth bounded by a function of k. The problem is then solved in the equivalent graph, using Courcelle's theorem [Courcelle 1990], by expressing the problem as an MSO predicate. Since we can express whether the weight of a subset of vertices is at most W using an MSO predicate of length bounded by a function of W, the following theorem follows from the results of Marx et al. [Marx 2013].

Theorem 2.6. *If $\mathcal{G}$ is hereditary and decidable, then $\mathcal{G}$-*Weighted Bipartition *is in* FPT.

Now we are ready to define the compression version of the problem Edge $(1,2)$-Partization and prove that it is in FPT. This implies that Edge $(1,2)$-Partization is in FPT.

Edge $(1,2)$-Partization Compression **Parameter:** k
Input: A Graph G with $V(G) = V \cup \{v\}$, an integer k and an edge set $S' \subseteq E(G-v)$, of size at most k, such that $G[V] - S'$ is a $(1,2)$-graph
Output: A subset $S \subseteq E$ of size at most k such that $G - S$ is a $(1,2)$-graph

Lemma 2.15. Edge $(1,2)$-Partization Compression *is in* FPT.

Proof. Let (G, k, S') be the input instance and $|V(G)| = n$. If $G - S'$ is a $(1,2)$-graph, then we return S' as the output. Otherwise we do the following. Let S be a *hypothetical solution* for the problem and let (P_I, P_C) be an IC-partition of $G - S$. Let $G' = G[V] - S'$. Since G' is a $(1,2)$-graph, the vertex set V can be partitioned into I, C_1 and C_2 such that (i) $G'[I]$ is a graph with no edges, and (ii) $G'[C_1]$ and $G'[C_2]$ are cliques. Since $G' = G[V] - S'$ and $I \subseteq V$ is a independent set in G', we know that $E(G[I]) \subseteq S'$. Also, since $|S'| \leq k$, we know that $|E(G[I])| \leq k$. Consider the partition of the vertex set of G, $V(G)$, into three parts I, $C_1 \cup \{v\}$ and C_2. Recall that (P_I, P_C) is an IC-partition of our hypothetical solution S. Our algorithm guesses the set of vertices $A = I \cap P_C$. Since P_C should be a complement of a bipartite graph, A should also be a complement of a bipartite graph. Hence, our algorithm guesses two cliques K_1 and K_2 from $G[I]$ and assumes that $A = K_1 \cup K_2$ will be part of P_C. By Lemma 2.13, the number of cliques in $G[I]$ is bounded by $2^{\mathcal{O}(\sqrt{k})}n$ and these cliques can be enumerated in time $2^{\mathcal{O}(\sqrt{k})}n$. After guessing A, we know that in our hypothetical IC-partition (P_I, P_C), $I \setminus A \subseteq P_I$ and $A \subseteq P_C$. Now, consider the partition (P'_I, P'_C) of $V(G)$, where $P'_I = I \setminus A$ and $P'_C = A \cup C_1 \cup C_2 \cup \{v\}$.

To solve the problem it is enough to find out a subset $U \subseteq C_1 \cup C_2 \cup \{v\}$ such that U is an OCT of the complement graph of $G[P'_C]$ and $|E(G[P'_I \cup U])| \leq k$. This can be encoded as a $\mathcal{G}$-Weighted Bipartition problem. Since $U \subseteq C_1 \cup C_2 \cup \{v\}$ and C_1 and C_2 are cliques, the cardinality of the set U will be bounded by $\mathcal{O}(\sqrt{k})$. The edges that contribute to $E(G[P'_I \cup U])$ are of three types–(i) edges within $G[P'_I]$, (ii) edges in $G[U]$ and (iii) edges between U and P'_I in G. Let $k_1 = |E(G[P'_I])|$. To encode the edges between U and P'_I we introduce a weight function w on P'_C. For each $u \in P'_C$, $w(u) = |N_G(u) \cap P'_I|$. Since we have fixed P'_I to be a subset of P_I, we need to include the set of edges in $E(G[P'_I])$ (type (i)) in the solution of the problem. The rest of the edges in the solution come from type (ii) or type (iii). Let $k_1 = |E(G[P'_I])|$, $k_2 = E(G[U])$ and k_3 be the number edges

between U and P'_I. So U is an OCT in the complement of $G[P'_C]$, of weight at most k_3 and number of edges in $G[U]$ is bounded by k_2.

So, our algorithm guesses the number of edges of type (ii) to be k_2, and type (iii) to be k_3. Let $\mathcal{G}_{k_2}$ be the class of graphs such that the number of edges in it is bounded by k_2. The class $\mathcal{G}_{k_2}$ is hereditary. To solve our problem it is enough to solve $\mathcal{G}_{k_2}$-Weighted Bipartition on the complement of the graph $G[P'_C]$ with weight function w. This completes the proof of the lemma. □

Thus by using Lemma 2.15, we get the following theorem.

Theorem 2.7. Edge $(1,2)$-Partization *is in* FPT.

2.7 Chapter Summary

In this chapter, we explored the parameterized complexity of a family of partition problems, namely Vertex (r,ℓ)-Partization and Edge (r,ℓ)-Partization. Except for Edge $(2,2)$-Partization, we concluded that Vertex (r,ℓ)-Partization and Edge (r,ℓ)-Partization is FPTwhen r or ℓ is at most 2. Whether there exists a polynomial kernel for the FPT problems remains an interesting open problem. Also, the parameterized complexity of Edge $(2,2)$-Partization remains unresolved.

Chapter 3

Parameterized Algorithms on Perfect Graphs for deletion to (r, ℓ)-graphs

3.1 Introduction

As mentioned earlier, many special subclasses of the (r, ℓ)-graph class, such as bipartite, chordal, interval, split and permutation, are well studied in various areas of algorithm design and intractability. Since a $(3, 0)$-graph is a 3-colourable graph, the recognition of an (r, ℓ)-graph, when either $r \geq 3$ or $\ell \geq 3$, is NP-complete [Garey 2002]. Thus, the following generalisation of Vertex (r, ℓ)-Partization is NP-hard when r or ℓ is at least 3:

Partization Recognition
Input: A graph G and positive integers r, ℓ
Question: Is G an (r, ℓ)-graph?

Partization Recognition has also been studied when the input is restricted to be a chordal graph. This restricted problem has a polynomial time algorithm [Feder 2011]. On the other hand, when the input graphs are restricted to perfect graphs, Partization Recognition is NP-complete [Wagner 1984]. It was shown in [Heggernes 2013], that the problem, when parameterized by $r + \ell$, has an FPT algorithm. A natural extension to Partization Recognition is the Vertex Partization problem. The problem is formally stated below:

Vertex Partization	**Parameter:** r, ℓ, k
Input: A graph G and positive integers r, ℓ, k	
Question: Is there a vertex subset $S \subseteq V(G)$ of size at most k such that $G - S$ is an (r, ℓ)-graph?	

As in the case of Partization Recognition, the Vertex Partization problem has also been studied when the input graph is restricted to a non-trivial graph class. By a result in [Addario-Berry 2010], this problem is NP-complete even when restricted to the perfect graph class. In fact, Odd Cycle Transversal (OCT) restricted to perfect graphs is NP-hard, because of this result. Thus, we can not expect an FPT algorithm for Vertex Partization parameterized by $r + \ell$, unless P = NP. Moreover, because of the NP-hardness of Partization Recognition on perfect graphs, we do not expect Vertex Partization on perfect graphs to be FPT when parameterized by k alone, again under the assumption that P $\neq$ NP. Krithika and Narayanaswamy [Krithika 2013]

studied Vertex Partization problems on perfect graphs, and among several results, they obtain an $(r+1)^k n^{\mathcal{O}(1)}$ algorithm for Vertex $(r,0)$-Partization on perfect graphs. In this chapter, we generalise this for all values of r and ℓ. In other words, we show that Vertex Partization on perfect graphs, parameterized by $k+r+\ell$, is FPT.

Our Results and Methods. For Vertex Partization on perfect graphs, parameterized by $k+r+\ell$, we give an FPT algorithm using the method of iterative compression. This algorithm is inspired by the FPT algorithm for Cochromatic Number on Perfect Graphs, given in [Heggernes 2013].

Then, we obtain a negative result for *kernelization* for Vertex Partization on perfect graphs. We show that Vertex Partization can not have a polynomial kernel unless NP $\subseteq$ coNP/poly. This is shown by exhibiting a polynomial parameter transformation from CNF-SAT. In fact, our result holds even when $k = 0$ and either r or ℓ is one. Thus, we show that Partization Recognition, parameterized by r, ℓ (also known as the Cochromatic Number problem [Heggernes 2013]), does not admit a polynomial kernel on perfect graphs unless NP $\subseteq$ coNP/poly. See Section 1.2 for the definition of polynomial parameter transformation and kernelization.

Finally, we consider the Vertex (r,ℓ)-Partization problem on perfect graphs. Recall that Vertex (r,ℓ)-Partization takes as input a graph G and a positive integer k, where k is the parameter. The aim is to determine whether there is a vertex subset $S \subseteq V(G)$ of size at most k, such that $G - S$ is an (r,ℓ)-graph. For each pair of constants r and ℓ, we give a polynomial kernelization algorithm for the above parameterized problem. To arrive at the kernelization algorithm, we again take help of the slightly larger class of (r,ℓ)-*split graphs*. Recall that a graph G is an (r,ℓ)-split graph if its vertex set can be partitioned into V_1 and V_2, such that the size of a largest clique in $G[V_1]$ is bounded by r and the size of a largest independent set in $G[V_2]$ is bounded by ℓ [Gyárfás 1998]. For any fixed r and ℓ, there is a finite forbidden set $\mathcal{F}_{r,\ell}$ for (r,ℓ)-split graphs [Gyárfás 1998]. That is, a graph G is an (r,ℓ)-split graph if and only if G does not contain any graph $H \in \mathcal{F}_{r,\ell}$ as an induced subgraph. In fact, since the input graph now is a perfect graph, we only need to consider forbidden perfect graphs. The size of the largest forbidden perfect graph is bounded by $g(r,\ell)$, for some function g depending only on r and ℓ [Kzdy 1996]. We use this to design the kernelization algorithm.

3.2 Preliminaries

A graph G is a *perfect graph* if, for every induced subgraph H, $\chi(H) = \omega(H)$. We also need the following characterization of perfect graphs – also known as strong perfect graph theorem.

Proposition 3.1 ([Chudnovsky 2006])**.** *A graph G is perfect if and only if G does not have, as an induced subgraph, an odd cycle of length at least 5 or its complement.*

This tells us that perfect graphs are closed under complementation. However, this was proved earlier and was called weak perfect graph theorem.

Lemma 3.1 ([Lovász 1972])**.** *G is a perfect graph if and only if $\overline{G}$ is a perfect graph.*

Moreover, this class is well known for its tractability for several NP-hard problems.

Proposition 3.2. ([Heggernes 2013, Lemma 3]). *Given a perfect graph G and an integer ℓ, there is a polynomial time algorithm to output*

(a) either a partition of $V(G)$ into at most ℓ independent sets or a clique of size $\ell+1$, and

(b) either a partition of $V(G)$ into at most ℓ cliques or an independent set of size $\ell+1$.

3.3 FPT algorithm for Vertex Partization

In this section, we show that Vertex Partization on perfect graphs is FPT, using the iterative compression technique. Let (G, r, ℓ, k) be an input instance of Vertex Partization on perfect graphs, and let $V(G) = \{v_1, \ldots, v_n\}$. We define, for every $1 \leq i \leq n$, the vertex set $V_i = \{v_1, \ldots, v_i\}$. Let G_i denote $G[V_i]$. Let $i_0 = r + \ell + k + 1$. We iterate through the instances (G_i, r, ℓ, k) starting from $i = i_0$. Given the i^{th} instance and a known (r, ℓ)-vertex deletion set S'_i of size at most $k+1$, our objective is to obtain an (r, ℓ)-vertex deletion set S_i of size at most k. The formal definition of this compression problem is as follows.

Vertex Partization Compression	**Parameter:** r, ℓ, k
Input: A perfect graph G, non-negative integers r, ℓ, k and a $k+1$-sized vertex subset S' such that $G - S'$ is an (r, ℓ)-graph, along with an IC-partition (Q_1, Q_2) of $G - S'$.	
Output: A vertex subset $S \subseteq V(G)$ of size at most k such that $G - S$ is an (r, ℓ)-graph, and an IC-partition (P_1, P_2) of $G - S$.	

Before we solve Vertex Partization Compression, we explain how to reduce the Vertex Partization problem to $n-(r+\ell+k+1)+1$ instances of the Vertex Partization Compression problem on G_i, from $i = i_0$ to $i = n$. For the graph G_{i_0}, the set V_{k+1} is a (r, ℓ)-vertex deletion set, S'_i, of size $k+1$. The graph $G_{i_0} - V_{k+1}$ has $r + \ell$ vertices. We set $Q_1^{i_0}$ to be a set of any r vertices of $V_{i_0} - V_{k+1}$ and $Q_2^{i_0}$ to be the remaining set of ℓ vertices; that is, $Q_2^{i_0} = V_{i_0} - V_{k+1} - Q_1^{i_0}$. Now, let $I_i = (G_i, r, \ell, k, S'_i, (Q_1^i, Q_2^i))$ be the i^{th} instance of Vertex Partization Compression. If S_{i-1} is a k-sized solution for I_{i-1}, then $S_{i-1} \cup \{v_i\}$ is a $(k+1)$-sized (r, ℓ)-vertex deletion set for G_i. So, the iteration begins with the instance $I_{i_0} = (G_{i_0}, r, \ell, k, V_{k+1}, (Q_1^{i_0}, Q_2^{i_0}))$ and we try to obtain an (r, ℓ)-vertex deletion set of size at most k. If such a solution S_{i_0} exists, we set $S'_{i_0+1} = S_{i_0} \cup \{v_{i_0+1}\}$ and ask for a k-sized solution for the instance I_{i_0+1}. We continue in this manner. If, during any iteration, the corresponding instance does not have an (r, ℓ)-vertex deletion set of size at most k, then this implies that the original instance (G, r, ℓ, k) is a NO instance for Vertex Partization. If S_n is a k-sized (r, ℓ)-vertex deletion set for G_n, where $G_n = G$, then clearly (G, r, ℓ, k) is YES instance of Vertex Partization. Since there are at most $n - (r + \ell + k + 1) + 1$ iterations, the total time taken by the algorithm to solve Vertex Partization is at most $n - (r + \ell + k + 1) + 1$ times the time taken to solve Vertex Partization Compression. Thus, if Vertex Partization Compression is FPT, it follows that Vertex Partization is FPT.

We can also *view* the input graph G of an instance of Vertex Partization Compression as an $(r + k + 1, \ell)$-graph with IC-partition $(Q_1 \cup S', Q_2)$. Equivalently, Vertex Partization Compression has as input positive integers r, ℓ, k and a graph G that is

an $(r+k+1, \ell)$-graph. The objective is to decide whether there is a k-sized set $S \subseteq V(G)$ such that $G - S$ is an (r, ℓ)-graph. This view point allows us to design an FPT algorithm for Vertex Partization Compression. First, we define some useful notations. Let G be a graph and $V(G) = \{v_1, \ldots, v_n\}$. For partition $P = (A, B)$ of $V(G)$ we define an n-length bit vector B_P^G corresponding to P as follows. We set the i^{th} bit to 0 if $v_i \in A$, and to 1 otherwise. For two n-length bit vectors $a = a_1 \ldots a_n$ and $b = b_1 \ldots b_n$, the hamming distance between a and b, denoted by $\mathcal{H}(a, b)$, is the number of indices on which a and b are mismatched. That is, $\mathcal{H}(a, b) = |\{(a_i, b_i) \mid a_i \neq b_i, i \in [n]\}|$. The Hamming distance for the bit vectors corresponding to two IC-partitions, of a graph, is bounded, as described by the following proposition.

Proposition 3.3 ([Heggernes 2013])**.** *Let G be a graph. Let Q be an IC-partition of G that realizes G as an (r', ℓ')-graph and P is an IC-partition of G that realizes G as an (r, ℓ)-graph. Then $\mathcal{H}(B_Q^G, B_P^G) \leq r'\ell + r\ell'$.*

The following lemma follows from Proposition 3.3.

Lemma 3.2. *Let G be a perfect graph and (Q_1, Q_2) be an IC-partition that realizes G as an (r', ℓ')-graph. Let S be a (r, ℓ)-vertex deletion set for G such that P is an IC-partition of G that realizes $G - S$ as an (r, ℓ)-graph and let $Q = (Q_1 \setminus S, Q_2 \setminus S)$. Then $\mathcal{H}(B_Q^{G-S}, B_P^{G-S}) \leq r'\ell + r\ell'$.*

Lemma 3.2 implies that to solve Vertex Partization Compression it is enough to solve the following problem.

Short Vertex Partization	**Parameter:** r, ℓ, k, ρ
Input: A perfect graph G, positive integers r, ℓ, k, ρ, and a partition $Q = (Q_1, Q_2)$ of $V(G)$.	
Output: A vertex subset $S \subseteq V(G)$ of size at most k such that $G - S$ is a (r, ℓ)-graph, and an IC-partition (P_1, P_2) of $G - S$ such that $\mathcal{H}(B_P^{G-S}, B_{Q'}^{G-S}) \leq \rho$ where $Q' = (Q_1 \setminus S, Q_2 \setminus S)$.	

Lemma 3.3. Short Vertex Partization *is* FPT.

Proof. We design a recursive algorithm $\mathcal{A}$ for Short Vertex Partization which takes a tuple $(G, r, \ell, k, \rho, Q = (Q_1, Q_2))$ as input, where G is a graph, Q is a partition of $V(G)$ and r, ℓ, k, ρ are integers. Suppose there exists a k-sized (r, ℓ)-vertex deletion set for the instance. Then $\mathcal{A}$ computes a k-sized (r, ℓ)-vertex deletion set S of G. Moreover, let P be an IC-partition that realizes $G - S$ as an (r, ℓ)-graph, such that $\mathcal{H}(B_P^{G-S}, B_{Q'}^{G-S}) \leq \rho$, where $Q' = (Q_1 \setminus S, Q_2 \setminus S)$. If P exists, then $\mathcal{A}$ returns the IC-partition P along with the set S. Otherwise, the algorithm returns NO. The following are the steps of the recursive algorithm $\mathcal{A}$ on input $(G, r, \ell, k, Q = (Q_1, Q_2), \rho)$.

1. If $k < 0$ or $\rho < 0$ then output NO.
2. If $G[Q_1]$ is r-colourable and $\overline{G}[Q_2]$ is ℓ-colourable, then return $(\emptyset, Q)$.
3. If $G[Q_1]$ is not r-colourable, then there is an $r+1$-sized clique in $G[Q_1]$. By Proposition 3.2, we can find such a $r+1$-sized clique C in polynomial time. Make the following recursive calls to $\mathcal{A}$:

(a) For every vertex $v \in V(C)$, do a recursive call $\mathcal{A}(G-v, r, \ell, k-1, \rho, (Q_1 \setminus \{v\}, Q_2 \setminus \{v\}))$ and if the recursive call returns (S', P) then return $(S' \cup \{v\}, P)$ as output.

(b) For every vertex $v \in C$, do a recursive call $\mathcal{A}(G, r, \ell, k, \rho-1, (Q_1 \setminus \{v\}, Q_2 \cup \{v\}))$ and if the recursive call returns (S', P) then return (S', P) as output.

If all the recursive calls in Step 3 return NO, then return NO.

4 If $\overline{G}[Q_2]$ is not ℓ-colourable, then there is clique of size $\ell+1$ in $\overline{G}[Q_2]$. Thus, using Proposition 3.2, we can find an $\ell+1$-sized independent set I in $G[Q_2]$. Make the following recursive calls of the algorithm:

(a) For every vertex $v \in I$, do a recursive call $\mathcal{A}(G-v, r, \ell, k-1, \rho, (Q_1 \setminus \{v\}, Q_2 \setminus \{v\}))$ and if the recursive call returns (S', P) then return $(S' \cup \{v\}, P)$ as output.

(b) For every vertex $v \in I$, do a recursive call $\mathcal{A}(G, r, \ell, k, \rho-1, (Q_1 \cup \{v\}, Q_2 \setminus \{v\}))$ and if the recursive call returns (S', P) then return (S', P) as output.

If all the recursive calls in Step 4 return NO, then return NO.

Correctness. We prove that the recursive algorithm $\mathcal{A}$ is correct. We show that if $(G, r, \ell, k, \rho, (Q_1, Q_2))$ is a YES instance of Short Vertex Partization, then the algorithm $\mathcal{A}$ will output the correct solution. We prove this by induction on $k+\rho$.

Base case: $k = 0$ and $\rho = 0$. Since $(G, r, \ell, k, \rho, (Q_1, Q_2))$ is a YES instance, (Q_1, Q_2) is an IC-partition which realizes that G is an (r, ℓ)-graph. In Step 2 of the algorithm, $\mathcal{A}$ correctly outputs $(\emptyset, (Q_1, Q_2))$.

Induction. By induction hypothesis, we assume that $\mathcal{A}$ outputs the correct answer when $k+\rho < \gamma$, where $\gamma \geq 0$. Now we show that $\mathcal{A}$ outputs the correct answer when $k+\rho = \gamma$. Let $(G, r, \ell, k, \rho, Q = (Q_1, Q_2))$ be the input of $\mathcal{A}$ such that $k+\rho = \gamma$. Let $(S, (P_1, P_2))$ be a solution of Short Vertex Partization. If $S = \emptyset$ and $(P_1, P_2) = (Q_1, Q_2)$ then in Step 2, algorithm $\mathcal{A}$ will output $(\emptyset, Q)$. Otherwise either $G[Q_1]$ is not r-colourable or $\overline{G}[Q_2]$ is not ℓ-colourable.

Case 1: Suppose $G[Q_1]$ is not r-colourable. Then there is a clique C of size $r+1$ in $G[Q_1]$. In this case at least one vertex v from C either belongs to S or does not belong to P_1. If $v \in S$, then consider the recursive call $\mathcal{A}(G-v, r, \ell, k-1, \rho, (Q_1 \setminus \{v\}, Q_2 \setminus \{v\})$. By induction hypothesis $\mathcal{A}(G-v, r, \ell, k-1, \rho, (Q_1 \setminus \{v\}, Q_2 \setminus \{v\})$ will return (S', P') such that S' is a $k-1$ sized (r, ℓ)-vertex deletion set of $G - \{v\}$ and $\mathcal{H}(B_{P'}^{G-S'}, B_{Q'}^{G-S'}) \leq \rho$, where $Q' = (Q_1 \setminus (S' \cup \{v\}), Q_2 \setminus (S' \cup \{v\}))$. Hence in Step 3$(a)$, algorithm $\mathcal{A}$ will output $(S' \cup \{v\}, P)$ and this is a solution for Short Vertex Partization on input $(G, r, \ell, k, \rho, (Q_1, Q_2))$.

If $v \notin S$, then $v \notin P_1$ as well. Consider the recursive call $\mathcal{A}(G, r, \ell, k, \rho-1, (Q_1 \setminus \{v\}, Q_2 \cup \{v\})$. By induction hypothesis, $\mathcal{A}(G, r, \ell, k, \rho-1, (Q_1 \setminus \{v\}, Q_2 \cup \{v\})$ will return (S'', P'') such that S'' is a k sized (r, ℓ)-vertex deletion set of G and $\mathcal{H}(B_{P''}^{G-S''}, B_{Q'}^{G-S''}) \leq \rho - 1$, where $Q' = ((Q_1 \setminus \{v\}) \setminus S'', (Q_2 \cup \{v\}) \setminus S'')$. Hence in Step 3$(b)$, algorithm $\mathcal{A}$ will output (S'', P''). Clearly S'' is a k sized (r, ℓ)-vertex deletion set of G''. Now we show that $\mathcal{H}(B_{P''}^{G-S''}, B_{Q''}^{G-S''}) \leq \rho$, where $Q'' = (Q_1 \setminus S'', Q_2 \setminus S'')$. Note that $\mathcal{H}(B_{Q'}^{G-S''}, B_{Q''}^{G-S''}) \leq 1$. Since $\mathcal{H}(B_{P''}^{G-S''}, B_{Q'}^{G-S''}) \leq \rho - 1$ and $\mathcal{H}(B_{Q'}^{G-S''}, B_{Q''}^{G-S''}) \leq 1$, it is only possible that $\mathcal{H}(B_{P''}^{G-S''}, B_{Q''}^{G-S''}) \leq \rho$

Case 2 : $\overline{G}[Q_2]$ is not ℓ-colourable. This case is symmetric to Case 1.

In the reverse direction, we show that if the algorithm $\mathcal{A}$ returns YES then indeed the given instance is a YES instance. The proof of the reverse direction is similar to the proof of forward direction. Again, we induct on $k + \rho$. The algorithm returns a bipartition as evidence of a YES instance. We show that this bipartition is the required IC-partition. Such a bipartition is returned in Steps 2, $3(a), (b)$ and $4(a), (b)$. By description of the algorithm, both integers $k, \rho \geq 0$ whenever the algorithm returns YES. In the base case, $k = \rho = 0$. Here, it must be the case that Step 2 is executed and the output bipartition is Q itself, while the output vertex deletion set is $\emptyset$. In this case, by definition Q is an IC-partition. Hence, Q is evidence that the input graph G is already an (r, ℓ)-graph and does not require any vertex to be deleted. Thus, in the base case, we correctly determine that the given input instance is a YES instance.

By induction hypothesis, we assume that $\mathcal{A}$ outputs the correct answer when $k + \rho < \gamma$, where $\gamma \geq 0$. We show that if $\mathcal{A}$ outputs a bipartition P and a vertex set S when $k+\rho = \gamma$, then the vertex set S is an (r, ℓ)-vertex deletion set of G, while the bipartition is an IC-partition of $G - S$. If the bipartition is output in Step 2, then by definition the input graph is already an (r, ℓ)-graph. Otherwise, a recursive call is made to an instance where $k + \rho$ is strictly lesser. By induction hypothesis, the recursive call returns an (r, ℓ)-vertex deletion set S' and a witnessing IC-partition of $G - S'$. It follows that the vertex set S and the bipartition output by the algorithm, on the current input instance, is an (r, ℓ)-vertex deletion set of size at most k, and an IC-partition respectively for the input instance. This completes the proof of correctness of the algorithm.

Running Time. Note that when $k < 0$ or $\rho < 0$, then the algorithm will stop in a single step. Each recursive call either decreases k by 1 or ρ by 1. Hence the depth of the recursion tree is bounded by $k + \rho + 1$. Note that in Step 3 we make at most $2(r + 1)$ recursive calls and in Step 4 we make at most $2(\ell + 1)$ recursive calls. Hence the total running time of the algorithm $\mathcal{A}$ is bounded by $\mathcal{O}(\max\{(2(r + 1))^{k+\rho+1} n^{\mathcal{O}(1)}, (2(\ell + 1))^{k+\rho+1} n^{\mathcal{O}(1)}\})$. □

Vertex Partization Compression is a special case of Short Vertex Partization when $\rho = (r + k + 1)\ell + r\ell$. Therefore, we have the following theorem.

Theorem 3.1. Vertex Partization *on perfect graphs has an* FPT *algorithm with running time* $2^{\mathcal{O}((k+r)\ell \log(r+\ell))} n^{\mathcal{O}(1)}$.

3.4 Kernel lower bound

In this section, we show that Vertex Partization on perfect graphs does not have an polynomial kernel. In fact, we show that Partization Recognition on perfect graphs can not have a polynomial kernel, when parameterized by $r + \ell$, unless NP $\subseteq$ coNP/poly. Partization Recognition on perfect graphs, when parameterized by $r + \ell$, was shown to be FPT in [Heggernes 2013].

Theorem 3.2. Partization Recognition *on perfect graphs, when parameterized by* $r + \ell$*, does not have a polynomial kernel unless* NP $\subseteq$ coNP/poly.

Proof. We prove the theorem by giving a polynomial parameter transformation from CNF-SAT parameterized by the number of variables. From Proposition 1.2, we know that

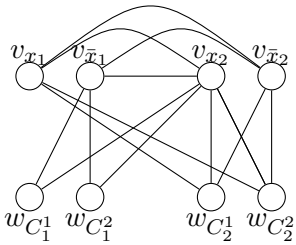


Figure 3.1: An illustration of the construction of the graph G in Theorem 3.2 for the formula $\phi = (x_1 \vee \overline{x}_2) \wedge (\overline{x}_1)$. Here $C_1 = (x_1 \vee \overline{x}_2)$ and $C_2 = (\overline{x}_1)$.

CNF-SAT, parameterized by the number of variables, does not have a polynomial kernel unless NP $\subseteq$ coNP/poly [Fortnow 2011]. Then the proof of the theorem follows from Proposition 1.1. We start with an instance (ϕ, n) of CNF-SAT, where ϕ is a CNF formula with m clauses and n variables. Without loss of generality, we assume that there is no clause where both literals of a variable appear together: such a clause will be satisfied by any assignment and can be removed. The polynomial parameter transformation produces an instance $(G, n, 1)$ of Partization Recognition, where G is a perfect graph, such that (ϕ, n) is a YES instance of CNF-SAT if and only if $(G, n, 1)$ is a YES instance of Partization Recognition. Let $\mathcal{C} = \{C_1, \ldots, C_m\}$, $X = \{x_1, \ldots, x_n\}$ and $L = \{x_1, \bar{x}_1, \ldots, x_n, \bar{x}_n\}$ be the set of clauses, variables and literals of ϕ respectively. The construction of the graph G from the formula ϕ is as follows (illustrated in Figure 3.1):

1. For each variable x, we create two vertices $v_x, v_{\bar{x}}$ which represent the literals $x, \bar{x}$. We call them the literal vertices. More specifically, we call v_x the positive literal vertex and $v_{\bar{x}}$ the negative literal vertex.

2. For each clause C, we create two vertices w_C^1, w_C^2. We call these the clause vertices corresponding to the clause C.

3. For each pair of variables x, y, we add the edges $v_x v_y, v_{\bar{x}} v_y, v_x v_{\bar{y}}$, and $v_{\bar{x}} v_{\bar{y}}$. Notice that $v_x v_{\bar{x}}$ and $v_y v_{\bar{y}}$ are non-edges.

4. For each clause C and each literal $q \in L$, if $q \notin C$, we add edges $x_q w_C^1$ and $x_q w_C^2$. In other words if a literal q' belongs to a clause C, then $q' w_C^1, q' w_C^2 \notin E(G)$ So, there is a complete bipartite graph between $L \setminus C$ and $\{w_C^1, w_C^2\}$.

In short, the vertex set and edge set of G is defined as follows (note that for a literal x, if $x = \bar{y}$, then $y = \bar{x}$).

$$\begin{aligned} V(G) &= \{v_x, v_{\bar{x}} \mid x \in X\} \cup \{w_C^1, w_C^2 \mid C \in \mathcal{C}\} \\ E(G) &= \{v_x v_y \mid x, y \in L, x \neq \bar{y}\} \cup \{w_C^1 v_x, w_C^2 v_x \mid x \in L, x \notin C, C \in \mathcal{C}\} \end{aligned}$$

Let $V_X = \{v_x, v_{\bar{x}} \mid x \in X\}$ and $V_{\mathcal{C}} = \{w_C^1, w_C^2 \mid C \in \mathcal{C}\}$. Note that the set of vertices $V_{\mathcal{C}}$, corresponding to the clauses, forms an independent set in G. First, we show that G is a perfect graph.

Claim 3.1. *The graph $\overline{G}$ does not contain an induced odd cycle of length ≥ 5.*

Proof. We first prove that there is no path of length 2 (path on 3 vertices) in $\overline{G}]V_X]$. Note that $E(\overline{G}[V_X]) = \{v_x v_{\bar{x}} \mid x \in X\}$. This implies that the degree of each vertex in the graph $\overline{G}[V_X]$ is exactly equal to 1. Hence, there is no path of length 2 in $\overline{G}[V_X]$. Also, since $V_{\mathcal{C}}$ forms a clique in $\overline{G}$, any induced cycle of length at least 5 in $\overline{G}$ will either contain a vertex or an edge from $V_{\mathcal{C}}$.

Let C' be an induced odd cycle of length at least 5 in $\overline{G}$. There are at most two vertices from $V_{\mathcal{C}}$ which are part of C'.Also, these vertices appear consecutively in C'. This implies that C' contains a path of length at least 2 using only vertices from V_X in $\overline{G}$, which is a contradiction. □

Claim 3.2. *The graph G does not contain an induced odd cycle of length ≥ 5.*

Proof. We first show that any induced odd cycle of length at least 5 can contain at most 3 vertices from V_X. Suppose not. Let C' be an induced odd cycle of length at least 5, such that $|V(C') \cap V_X| \geq 4$. Let v_w, v_x, v_y, v_z be four distinct vertices from $V(C') \cap V_X$, appearing in that order if we go around the cycle in a clockwise manner. That is, there are paths $v_w \cdots v_x$, $v_x \cdots v_y$, $v_y \cdots v_z$ in C'. Since C' is an induced cycle, there is no edge $v_w v_y$ in $E(G)$. This implies that $y = \bar{w}$. By similar arguments, we can show that $x = \bar{z}$. This implies that $v_w v_x v_y v_z v_w$ form a cycle of length 4 in G, contradicting the fact that C' is an induced odd cycle containing v_w, v_x, v_y and v_z. Hence, any induced odd cycle of length at least 5 can contain at most 3 vertices from V_X.

Since $V_{\mathcal{C}}$ is an independent set in G, no two vertices from $V_{\mathcal{C}}$ can occur as consecutive vertices in any cycle. Let C' be an induced odd cycle of length at least 5 in G. Since $|V(C') \cap V_X| \leq 3$ and no two vertices from $V_{\mathcal{C}}$ appear as consecutive vertices in C', it must be the case that $|V(C') \cap V_{\mathcal{C}}| \leq 2$. This implies that the length of C' is exactly equal to 5 and C' is of the form $v_x w^i_{C_1} v_y w^j_{C_2} v_z v_x$, where $i, j \in \{1, 2\}$. Since C' is an induced cycle $v_x v_y, v_y v_z \notin E(G)$. This implies that $y = \bar{x} = z$ and hence $v_y = v_z$. This contradicts the fact that C' is a cycle. This completes the proof of the claim. □

Proposition 3.1 and Claims 3.1 and 3.2 imply that G is a perfect graph. We now show that (ϕ, n) is a YES instance of CNF-SAT if and only if $(G, n, 1)$ is a YES instance of Partization Recognition.

First, suppose that (ϕ, n) is a YES instance of CNF-SAT. Then, there is an assignment τ, such that every clause has at least one literal set to 1. Let $f : \mathcal{C} \to X$ be a map that arbitrarily maps one such satisfying literal to each clause. Note that for a clause C, $w^1_C v_{f(C)}, w^2_C v_{f(C)} \notin E(G)$, because $f(C) \in C$. We construct n independent sets as follows. For each literal y, if $\tau(y) = 1$, let $I_y = \{w^1_C, w^2_C \mid f(C) = y\} \cup \{v_y\}$. Since $V_{\mathcal{C}}$ is an independent set $I_y \setminus \{v_y\}$ is an independent set. Note that for all $w^i_C \in I_y$, $i \in \{1, 2\}$ we have that $w^i_C v_y \notin E(G)$, because $f(C) = y$ and $y \in C$. This implies that I_y is an independent set. Since τ is an assignment, exactly one of the literals of each variable is assigned 1 by τ. Thus, in this way we form n independent sets. Since τ is a satisfying assignment for ϕ, the function f maps each clause C to a literal which is assigned 1 by τ. This implies that all vertices in $V_{\mathcal{C}}$ are covered by the independent sets constructed above. The vertices in the graph G, which are not covered by the independent sets constructed, correspond to the literals that have been set to 0 by τ. By construction of G and by the definition of an assignment τ, these vertices form a clique. Hence, $(G, n, 1)$ is an YES instance of Partization Recognition.

Conversely, suppose $(G, n, 1)$ is a YES instance of Partization Recognition. Then there is an (r, ℓ)-partition $\mathcal{P}$ of G. Let $I_1, \ldots, I_n$ be n independent sets and K be a clique in the (r, ℓ)-partition $\mathcal{P}$. It is not possible, by construction of G, that there is a variable x such that both v_x and $v_{\bar{x}}$ belong to the clique K, because $v_x v_{\bar{x}} \notin E(G)$. As there is only one clique in $\mathcal{P}$, at most one literal of each variable can be contained in the clique K of $\mathcal{P}$. Hence, for each variable x, either v_x or $v_{\bar{x}}$ is part of an independent set in $\mathcal{P}$. Furthermore, since for two literals p and q with $p \neq \bar{q}$, $v_p v_q \in E(G)$, any independent set I in $\mathcal{P}$ can not contain both v_p and v_q. This implies that each of the n independent sets can be identified by a variable $x \in X$. Since there are only n independent sets in $\mathcal{P}$, there can not be a variable x such that both v_x and $v_{\bar{x}}$ are part of distinct independent sets in $\mathcal{P}$. Thus, the construction of G forces only two possibilities for each variable:

(a) there is exactly one literal vertex that is part of an independent set while the other one belongs to the clique K, or

(b) both literals together form an independent set, which has no other vertices of G. This is because of the assumption that for each variable x there is no clause containing both x and $\bar{x}$.

We construct an assignment τ and show that τ is a satisfying assignment for ϕ. For a literal z, if $v_z \in K$, then we set $\tau(z) = 0$. If for a variable x, both vertices v_x and $v_{\bar{x}}$ do not belong to K then we set $\tau(x) = 1$. We show that τ is a satisfying assignment for ϕ. Let C be a clause in the formula ϕ. Since $w_C^1 w_C^2 \notin E(G)$, at least one of w_C^1 or w_C^2 belongs to an independent set I in $\mathcal{P}$. Let $w_C^i \in I$, where $i \in \{1, 2\}$. We have shown that each independent set contains at least one vertex corresponding to a literal y. Since v_y and w_C^i belong to I, it must be that $v_y w_C^i \notin E(G)$. This implies that $y \in C$. Furthermore, this implies that $v_{\bar{y}} \notin I$ as no clause contains both y and $\bar{y}$. Hence, $v_{\bar{y}}$ is in K and $\tau(\bar{y}) = 0$. This implies that y is set to 1 and hence the clause C is satisfied. This proves that τ is a satisfying assignment for ϕ. □

Partization Recognition is same as Vertex Partization, when $k = 0$. Hence we get the following corollary.

Corollary 3.1. Vertex Partization *parameterized by $k + r + \ell$ on perfect graphs does not have a polynomial kernel unless* NP $\subseteq$ coNP/poly.

3.5 Polynomial kernel when r and ℓ are constants

We saw that there is no polynomial kernel for Vertex Partization, unless NP $\subseteq$ coNP/poly. The parameter for this problem is $k + r + \ell$, where the size of the deletion set is at most k and the final graph is an (r, ℓ)-graph. In this section, we consider the Vertex (r, ℓ) Partization problem on perfect graphs, which is a special case of Vertex Partization on perfect graphs. Here, for a given pair of fixed positive constants (r, ℓ), we take a perfect graph G and a positive integer k as input and decide whether there is a vertex set S of size at most k the deletion of which results in an (r, ℓ)-graph. This simplified problem has a polynomial kernel, as shown below.

We first observe that when perfect graphs are (r, ℓ)-graphs, this class coincides with another graph class, namely the class of perfect graphs that are (r, ℓ)-split graphs. From the definition of the graph classes, it is true that any (r, ℓ)-graph is also an (r, ℓ)-split graph.

Lemma 3.4. *Let G be a perfect graph. If G is an (r, ℓ)-split graph, then G is also an (r, ℓ)-graph.*

Proof. Since G is a perfect graph, for any induced subgraph G' of G, the chromatic number of G' ($\chi(G')$) is equal to the maximum size of a clique of G' ($\omega(G')$). We know that G is an (r, ℓ)-split graph. Let (P_1, P_2) be an (r, ℓ)-split partition with $\omega(G[P_1]) \leq r$ and $\alpha(G[P_2]) \leq \ell$ Now we show that (P_1, P_2) is indeed an (r, ℓ)-partition of G. Since G is a perfect graph, the graphs $G[P_1]$ and $\overline{G}[P_2]$ are perfect graphs. Since $G[P_1]$ is a perfect graph and $w(G[P_1]) \leq r$, $\chi(G[P_1]) \leq r$. This implies that P_1 can be partitioned into r independent sets. Since $\overline{G}[P_2]$ is a perfect graph and $\alpha(G[P_2]) \leq \ell$, $w(\overline{G}[P_2]) \leq \ell$ and hence $\chi(\overline{G}[P_2]) \leq \ell$. This implies that P_2 can be partitioned into ℓ sets such that each set is independent in $\overline{G}[P_2]$. Hence P_2 can be partitioned into ℓ cliques in $G[P_2]$. So (P_1, P_2) is an (r, ℓ)-partition of G. This completes the proof of the lemma. □

For any fixed r and ℓ, there is a finite forbidden set $\mathcal{F}'_{r,\ell}$ of perfect graphs for (r, ℓ)-split graphs [Kzdy 1996]. In other words, a perfect graph G is an (r, ℓ)-split graph if and only if G does not contain any graph $H \in \mathcal{F}_{r,\ell}$ as an induced subgraph. The size of the largest forbidden graph is bounded by $g(r, \ell) \leq 2(\ell + 1)(R(r(\ell + 1), (r(s + 1))^2 + r(s + 1) + 3) + 1)$ [Kzdy 1996]. Since $g(r, \ell)$ is a constant for constant r, ℓ, it is possible to compute the forbidden set $\mathcal{F}'_{r,\ell}$ in polynomial time. Thus, the class of perfect (r, ℓ)-graphs also has a finite forbidden characterization. This implies that Vertex (r, ℓ) Partization on perfect graphs reduces to the d-Hitting Set problem, where d is the constant $g(r, \ell)$. In an equivalent d-Hitting Set instance, the universe will be the set of vertices of the input graph G, while the family of sets will be the vertex sets of induced subgraphs of G that are isomorphic to a forbidden graph. The set sizes are bounded by $g(r, \ell)$. Hence, by [Abu-Khzam 2010], this problem has a polynomial kernel. This gives us the following theorem.

Theorem 3.3. Vertex (r, ℓ) Partization *on perfect graphs admits a kernel of size $k^{\mathcal{O}(d)}$ and has an algorithm with running time $d^k n^{\mathcal{O}(d)}$. Here, $d = g(r, \ell)$.*

3.6 Chapter Summary

In this chapter, we studied the Vertex Partization problem on perfect graphs, and showed that it is FPT but does not admit a polynomial kernel. Furthermore, we observed that Vertex (r, ℓ) Partization has an induced finite forbidden characterization and utilized that to give a polynomial kernel for the problem. However, this preprocessing takes $n^{\mathcal{O}(d)}$ time, where d depends on the size of a largest graph in the finite forbidden set. It would be interesting to replace the factor $n^{\mathcal{O}(d)}$ by $\tau(d) \cdot n^{\mathcal{O}(1)}$.

Chapter 4

Communication Complexity of Separating Families with Applications

4.1 Introduction

The two party communication complexity, introduced by Yao [Yao 1979], is an important research area in theoretical computer science with many applications. This notion of complexity is particularly useful for proving lower bounds for VLSI computation, parallel computation, data structures as well as circuit lower bounds. In this model of communication, there are two players, Alice and Bob, holding inputs $x \in X$ and $y \in Y$ respectively, and they want to compute a given function $f : X \times Y \to \{0,1\}$, by communicating as few bits as possible. The minimum number of bits communicated, for any pair of inputs (x, y), to compute the function f, is called the (deterministic) communication complexity of f, denoted by $D(f)$. One such communication complexity problem, which has garnered a lot of attention, is the Clique vs Independent Set problem, introduced by Yannakakis [Yannakakis 1991]. For an n-vertex graph G, the Clique vs Independent Set problem is defined as follows. Alice gets a clique C in G and Bob gets an independent set I in G. Here both Alice and Bob know the graph G and their goal is to decide whether the clique and the independent set intersect at any vertex, by exchanging as few bits as possible. In other words, define the function $\mathsf{CIS}_G(C, I)$ as the cardinality of $V(C) \cap V(I)$ (note that $|V(C) \cap V(I)| \in \{0, 1\}$) and, Alice and Bob want to compute $\mathsf{CIS}_G(C, I)$. It can be shown that $D(\mathsf{CIS}_G) = \mathcal{O}(\log^2 n)$. One can also show that $D(\mathsf{CIS}_G) = \Omega(\log n)$, using the fooling set technique, a method to show communication lower bounds. Closing the gap between the upper and lower bound of CIS_G is a long standing open problem. Very recently, in 2015, Göös et al. [Göös 2015c] showed a near optimal lower bound of $\tilde{\Omega}(\log^2 n)$ for the problem, where $\tilde{\Omega}(m)$ hides factors poly-logarithmic in m. Later, Göös et al. [Göös 2015b] showed that the same lower bound holds even for randomized communication complexity of the problem. Other versions of two party communication protocols deal with the concepts of nondeterministic and co-nondeterministic protocols. There are many works which study the cost of co-nondeterministic communication protocols of the Clique vs Independent Set problem [Huang 2012, Amano 2014, Shigeta 2015, Göös 2015a]. For more details on nondeterministic, co-nondeterministic and randomized communication complexities, please refer to [Lovász 1990].

In this chapter, we study the communication complexity of graph properties that generalise the function CIS_G. Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be two hereditary graph properties. That is, $\mathcal{F}_1$ and $\mathcal{F}_2$ are two families of graphs such that if $G \in \mathcal{F}_i, i \in \{1, 2\}$, then all induced subgraphs of G are also in $\mathcal{F}_i$. We define a $(\mathcal{F}_1, \mathcal{F}_2)$ communication problem as follows. For any fixed n-

vertex graph G, Alice gets an induced subgraph G_1 of G and Bob gets an induced subgraph G_2 of G, such that $G_i \in \mathcal{F}_i, i \in \{1,2\}$, and their objective is to check whether $V(G_1)$ and $V(G_2)$ intersect, by communicating as few bits as possible. In other words, we define a function $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$ as $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}(G_1,G_2) = 1$ if $V(G_1)$ and $V(G_2)$ do not intersect and 0 otherwise, where G_1 and G_2 are induced subgraphs of G and $G_i \in \mathcal{F}_i, i \in \{1,2\}$. Alice and Bob want to find the value of the function $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$ on (G_1,G_2). Notice that, when $\mathcal{F}_1$ is the family of cliques and $\mathcal{F}_2$ is the family of independent sets, then $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}(C,I) = 1$ if and only if $\mathsf{CIS}_G(C,I) = 0$. A trivial protocol for computing $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$ is as follows: Alice sends a bit vector of $V(G_1)$ to Bob and Bob checks whether it intersects with the vertex set $V(G_2)$; the number of bits communicated in this protocol is n. One of our main theorems characterizes pairs of graph families for which the trivial protocol is the best one.

Theorem 4.1. *For any two hereditary families of graphs $\mathcal{F}_1$ and $\mathcal{F}_2$, for any integer $n > 0$, there is an n-vertex graph G such that $D(\mathsf{GDIS}_{G,\mathcal{F}_1,\mathcal{F}_2}) = \Omega(n)$ if and only if $\mathcal{F}_1 \cap \mathcal{F}_2$ is an infinite family.*

We give a sketch of the proof for this Theorem. We observe that when a pair of hereditary graph families have finitely many graphs in their intersection, they have the following property: In one family, all graphs have their independence number (the maximum size of an independent set) bounded by a constant, while in the other family, all graphs have their clique number (the maximum size of a clique) bounded by some other constant. Thus, we consider the communication complexity for computing $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$ when $\mathcal{F}_1$ and $\mathcal{F}_2$ are specific families. Let $\mathcal{C}_r$ be the family of graphs such that the independence number is at most r, and $\mathcal{I}_\ell$ be the family of graphs such that the clique number is at most ℓ. Such pairs of families were considered in the study made in [Feder 1999]. We are able to show that $D(\mathsf{GDISJ}_{G,\mathcal{C}_r,\mathcal{I}_\ell}) = \mathcal{O}(\log^{r+\ell} n)$ and, therefore, conclude the hypothesis of Theorem 4.1.

One of our main motivation, to carry out this study, was to introduce ideas from parameterized complexity in the study of communication complexity. Parameterized complexity theory is a framework for a refined analysis of primarily hard (NP-hard) problems. Here, every input instance I of a problem Π is accompanied with an integer parameter k, and the running time is measured in terms of the associated parameter k and the input size. The main idea of parameterized algorithms is to *measure* the running time in terms of both input size as well as a parameter that captures structural properties of the input instance. Using ideas from parameterized complexity, we obtain the following nuanced upper bounds on the communication complexity of the function $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$. A pair $(\mathcal{F}_1,\mathcal{F}_2)$ of hereditary graph families where $\mathcal{F}_1 \cap \mathcal{F}_2$ is a finite, will be referred to as a *good pair of graph families.*

Theorem 4.2. *Let G be an n-vertex graph and $(\mathcal{F}_1,\mathcal{F}_2)$ be a good pair of graph families. Let $\mathsf{opt}^G_{\mathcal{F}_1,\mathcal{F}_2}$ be the size of a minimum set S of vertices such that $V(G)\setminus S$ has a bipartition (V_1,V_2) with $G[V_1] \in \mathcal{F}_1$ and $G[V_2] \in \mathcal{F}_2$. Then there is a protocol for $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$ that has $\mathcal{O}(\log^c(\mathsf{opt}^G_{\mathcal{F}_1,\mathcal{F}_2}) + \log n)$ communication complexity. Here, c is a constant depending only on the two graph families.*

For the special case of Clique vs Independent Set problem we get a protocol that has $\mathcal{O}(\log^2(\mathsf{opt}^G_{\mathcal{C}_1,\mathcal{I}_1}) + \log n)$ communication complexity. We also study communication complexity in terms of the degeneracy of graphs in the given family. In particular, we consider the pair of families $(\mathcal{C}_1,\mathcal{D}_\ell)$, where $\mathcal{D}_\ell$ is the set of all ℓ-degenerate graphs and

$\overline{\mathcal{C}}_1$ is the set of all complete graphs. As mentioned in Chapter 1.2, an ℓ-degenerate graph is an undirected graph where every induced subgraph has a vertex with degree at most ℓ. Note that $\mathcal{D}_0$ is the family of independent sets. Hence, this is still a generalisation of the Clique vs Independent Set problem. We prove the following theorem regarding the communication complexity of $\mathsf{GDISJ}_{G,\mathcal{C}_1,\mathcal{D}_\ell}$.

Theorem 4.3. *For any constant $\ell \in \mathbb{N}$, $D(\mathsf{GDISJ}_{G,\mathcal{C}_1,\mathcal{D}_\ell}) = \mathcal{O}(\ell \log^2 n)$, where $n = |V(G)|$.*

Separating Families. The main motivation for Yannakakis to introduce the Clique vs Independent Set problem was to study the number of constraints in the linear programming of a vertex packing polytope. As a spin-off of this study, he provided relations between the Clique vs Independent set problem and a CI-separating family (Clique-Independent set separating family): for a graph G, a family $\mathcal{F}$, of subsets of $V(G)$, is called a CI-separating family if for any disjoint clique C and independent set I in G, there is a set $F \in \mathcal{F}$ such that $C \subseteq F$ and $I \cap F = \emptyset$. He showed that the co-nondeterministic communication complexity of CIS_G is $\log q(G)$, where $q(G)$ is the cardinality of a CI-separating family of G. Yannakakis also provided a polynomial sized CI-separating family on comparability graphs and their complements, chordal graphs and their complements, and asked whether there is a polynomial sized family on general graphs, or even on perfect graphs. Lovász [Lovász 1994] extended the work of Yannakakis to t-perfect graphs and gave a polynomial sized CI-separating family on t-perfect graphs. Bousquet et al. [Bousquet 2014] proved the existence of polynomial sized CI-separating families for the following class of graphs: random graphs, split-free graphs (here the graph does not have a fixed split graph as an induced subgraph), graphs with no induced path P_k on k vertices nor its complement (here k is a constant), and graphs with no induced P_5. But, a result of Göös [Göös 2015a], that shows that the co-nondeterministic communication complexity of CIS_G is $\Omega(\log^{1.128} n)$, implies that the cardinality of CI-separating family on general graphs is super polynomial in the number of vertices.

The communication complexity of CIS_G, $D(\mathsf{CIS}_G) = \mathcal{O}(\log^2 n)$ implies that there exists a CI-separating family of cardinality $n^{\mathcal{O}(\log n)}$ (See [Lovász 1990]). We would like to remark that the existence of a CI-separating family does not imply that such a family can be enumerated in time polynomial in the size of the family. The best known bound on the cardinality of a enumerable CI-separating family on general graphs is $\mathcal{O}(n^{\frac{\log n}{2}})$, by Hajnal (unpublished, cited in [Lovász 1994]). Cygan et. al. [Cygan 2013] also enumerated a CI-separating family, of cardinality $n^{\mathcal{O}(\log n)}$, in time $n^{\mathcal{O}(\log n)}$. In the special case of finding a CI-separting family, one can use the communication protocol of CIS_G, given in [Lovász 1994], to enumerate such a family in time $n^{\mathcal{O}(\log n)}$. To generalise from the definition of CI-separating families, for a graph G, and a pair of families $\mathcal{F}_1$ and $\mathcal{F}_2$, a notion of $(\mathcal{F}_1, \mathcal{F}_2)$-separating family was introduced. A family $\mathcal{P}$ of vertex subsets of $V(G)$ is called a $(\mathcal{F}_1, \mathcal{F}_2)$-separating family if for any two disjoint vertex subsets V_1 and V_2 with $G[V_1] \in \mathcal{F}_1$ and $G[V_2] \in \mathcal{F}_2$, there is a set $A \in \mathcal{P}$ such that $V_1 \subseteq A$ and $V_2 \cap A = \emptyset$.

From an observation made in [Lovász 1990], it is implied that a non-deterministic protocol for $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$ corresponds to a $(\mathcal{F}_1, \mathcal{F}_2)$-separating family. This means that if $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$ has deterministic communication complexity c, then there exists a $(\mathcal{F}_1, \mathcal{F}_2)$-separating family of size 2^c. Similar to the case of CI-separating families, describing an enumeration algorithm to find the best separating family is a problem of wide interest. We show that a $(\mathcal{C}_r, \mathcal{I}_\ell)$-separating family of size $2^{\mathcal{O}(\log^{r+\ell} n)}$ can be enumerated in time $2^{\mathcal{O}(\log^{r+\ell} n)}$. This, in turn, implies the following theorem.

Theorem 4.4. *For any two hereditary families of graphs $\mathcal{F}_1$ and $\mathcal{F}_2$, for each integer $n > 0$, there is an n-vertex graph G such that any $(\mathcal{F}_1, \mathcal{F}_2)$-separating family must be of size $2^{\Omega(n)}$ if and only if $\mathcal{F}_1 \cap \mathcal{F}_2$ is an infinite family.*

We get the following theorem as a "separating family" analogue of Theorem 4.2. This theorem is extremely useful in designing parameterized algorithms.

Theorem 4.5. *Let $(\mathcal{F}_1, \mathcal{F}_2)$ be a good pair of graph families. Given an n-vertex graph G, let S be a minimum sized vertex set such that $V(G) \setminus S$ has a bipartition (V_1, V_2) with $G[V_1] \in \mathcal{F}_1$ and $G[V_2] \in \mathcal{F}_2$. Let $|S| = \mathsf{opt}^G_{\mathcal{F}_1,\mathcal{F}_2}$. Then a $(\mathcal{F}_1, \mathcal{F}_2)$-separating family, for G, of cardinality $2^{\mathcal{O}(\log^c \mathsf{opt}^G_{\mathcal{F}_1,\mathcal{F}_2})} n^{\mathcal{O}(1)}$ can be enumerated in time $2^{\mathcal{O}(\log^c \mathsf{opt}^G_{\mathcal{F}_1,\mathcal{F}_2})} n^{\mathcal{O}(1)}$, where c is a constant.*

Another pair of graph properties (families of graphs) we consider is the family of complete graphs, $\mathcal{C}_1$ and that of ℓ-degenerate graphs, $\mathcal{D}_\ell$. By Theorem 4.3, we already know that the communication complexity of computing $\mathsf{GDIS}_{G,\mathcal{C}_1,\mathcal{D}_\ell}$ is $\mathcal{O}(\ell \log^2 n)$. We also give an algorithm to enumerate a $(\mathcal{C}_1, \mathcal{D}_\ell)$-separating family for an n-vertex graph, of cardinality $n^{\mathcal{O}(\ell \log n)}$, in time $n^{\mathcal{O}(\ell \log n)}$.

Applications. In 2013, Cygan et al. [Cygan 2013] drew a very interesting relation between the field of enumerating separating families and designing algorithms. As mentioned earlier, a CI-separating family of cardinality $n^{\mathcal{O}(\log n)}$ is enumerated in time $n^{\mathcal{O}(\log n)}$, and this family is used to design fast exact and parameterized algorithms. They showed that Split Vertex Deletion, where we want to delete at most k vertices from a given n-vertex graph to get a split graph, can be solved in time $\mathcal{O}(1.2738^k k^{\mathcal{O}(\log k)} + n^3)$. They also showed that all induced split subgraphs of a given n-vertex graph can be listed in time $\mathcal{O}(3^{n/3} n^{\mathcal{O}(\log n)})$ time. This work motivated the last part of our study: designing exact and FPT algorithms. Not only are the enumeration algorithms for separating families interesting combinatorial questions in their own right, but they also help to design fast FPT and exact exponential time algorithms for a class of problems. A generic class of problems for which a separating family based approach works is as follows. Let $\mathcal{G}$ be a family of graphs. Then $\mathcal{G} + kv$ contains all graphs G such that there is a vertex set $S \subseteq V(G)$, of size at most k, with the property that the graph $G \setminus S \in \mathcal{G}$. Given two graph families $\mathcal{F}_1, \mathcal{F}_2$, we consider the following problem.

$(\mathcal{F}_1, \mathcal{F}_2)$-p-Partition	**Parameter:** k
Input: A graph G, a non-negative integer k	
Question: Is there a vertex set $S \subseteq V(G)$, of size at most k, such that there is a partition $V_1 \uplus V_2$ of $V(G) \setminus S$ and $G[V_i] \in \mathcal{F}_i$, $i \in \{1, 2\}$?	

The optimization version of $(\mathcal{F}_1, \mathcal{F}_2)$-p-Partition is denoted by $(\mathcal{F}_1, \mathcal{F}_2)$-Partition. Here, the aim is to find the minimum size of a vertex set S such that $V(G) \setminus S$ has a bipartition (V_1, V_2) with $G[V_1] \in \mathcal{F}_1$ and $G[V_2] \in \mathcal{F}_2$. Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be a good pair of graph families. For any positive integer k, let the families $\mathcal{F}_1 + kv$ and $\mathcal{F}_2 + kv$ have FPT recognition algorithms. That is, there are algorithms which take as input a graph G and an integer k, decide whether $G \in \mathcal{F}_i + kv, i \in \{1, 2\}$ and run in time $f(k)|V(G)|^{\mathcal{O}(1)}$. For ease of notation, if $\mathcal{F}_1$ and $\mathcal{F}_2$ be a good pair of graph families, and the families $\mathcal{F}_1 + kv$ and $\mathcal{F}_2 + kv$ have FPT recognition algorithms, then we call $(\mathcal{F}_1, \mathcal{F}_2)$ an FPT-*good pair of families*.

Theorem 4.6. *Let $(\mathcal{F}_1, \mathcal{F}_2)$ be an* FPT*-good pair of families. Also, let $\mathcal{A}_1$ and $\mathcal{A}_2$ be the best recognition algorithms for $\mathcal{F}_1 + kv$ and $\mathcal{F}_2 + kv$ respectively. For an n-vertex input graph and non-negative integer k, let the running time of $\mathcal{A}_i, i \in \{1,2\}$, be $T_i(n,k)$. Then, $(\mathcal{F}_1, \mathcal{F}_2)$-*p-Partition*, on an instance (G,k), can be solved in time $2^{\mathcal{O}(\log^c k)} n^{\mathcal{O}(1)} \cdot \max\{T_1(n,k), T_2(n,k)\}$.*

One could obtain a result analogous to Theorem 4.6 for $(\mathcal{F}_1, \mathcal{F}_2)$-Partition. A few of the problems for which we obtain faster FPT and exact algorithms are (Clique,Planar)-p-Partition, (Clique,Triangle-free)-p-Partition, (Clique,Forest)-p-Partition and (Clique, Treewidth-t)-p-Partition. We also obtain an improved algorithm for Vertex (r,ℓ)-Partization, when $1 \leq r, \ell \leq 2$. In Chapter 2, we gave an FPT algorithm running in $3.3146^k |V(G)|^{\mathcal{O}(1)}$ time, where (G,k) was the input instance (Theorem 2.1 and Corollary 2.1). In this chapter, we obtain an algorithm running in time $2.3146^k |V(G)|^{\mathcal{O}(1)}$, which is as good an algorithm as the best known algorithm for OCT. This means that the results of Table 2.1 now stand as given in Table 4.1.

Table 4.1: Updated summary of known and new results for the family of Vertex (r,ℓ)-Partization problems. New results are highlighted in green (last row).

r,ℓ	Problem Name	FPT	Kernel
$(1,0)$	Vertex Cover	1.2738^k	Poly
$(0,1)$	Vertex Cover on $\overline{G}$	1.2738^k	Poly
$(1,1)$	SVD	$1.2738^{k+o(k)}$	Poly
$(2,0)$	OCT	2.3146^k	Randomized Poly
$(0,2)$	OCT on $\overline{G}$	2.3146^k	Randomized Poly
$(2,1)$, $(1,2)$, $(2,2)$	Vertex $(2,1)$-partization Vertex $(1,2)$-partization Vertex $(2,2)$-partization	2.3146^k	**Randomized Turing Poly**

4.2 Preliminaries

We give a formal definition of a communication protocol.

Definition 4.1 ([Kushilevitz 1997])**.** *A protocol Π over a domain $X \times Y$ with range Z is a binary tree where each internal node v is labelled either by a function $a_v : X \to \{0,1\}$ or by a function $b_v : Y \to \{0,1\}$, and each leaf is labelled with an element $z \in Z$. The value of the protocol Π on an input (x,y) is the label of the leaf reached by starting at the root, and walking along a path in the tree. At each internal node v labelled by a_v, the walk takes left if $a_v(x) = 0$ and right if $a_v(x) = 1$, and at each internal node labelled by b_v, the walk takes left if $b_v(y) = 0$ and right if $b_v(y) = 1$. The cost of the protocol Π on an input (x,y) is the length of the path taken on the input (x,y). The cost of the protocol Π is the height of the binary tree.*

Informally, a protocol can be thought of as a communication between two players, Alice and Bob. They have decided on some function f and wish to evaluate $f(x,y)$, for some input $x \in X$ and $y \in Y$. The catch is that x is only known to Alice and y is only known to Bob. Thus, they fix a protocol of communication, such that at the end $f(x,y)$ is known

to both of them. Therefore, for each internal node v of the binary protocol tree, the label is a_v or b_v according to which player is communicating at that point of the protocol. The value $a_v(x)$ is the bit that Alice sends to Bob. Similarly, b_v corresponds to the bit sent by Bob to Alice.

Definition 4.2 ([Kushilevitz 1997])**.** *For a function $f : X \times Y \to Z$, the deterministic communication complexity of f is the minimum cost of Π, over all protocols Π that compute f. We denote the deterministic communication complexity of f by $D(f)$.*

For further reading on Communication Complexity, including the concepts of non-deterministic and co-nondeterministic communication complexity of a function, we refer the reader to [Kushilevitz 1997, Lovász 1990]. One of the first functions, whose communication complexity was studied, is the DISJOINTNESS function. For any $x, y \in \{0,1\}^n$, the function is defined as,

$$\mathsf{DISJ}_n(x,y) = \begin{cases} 0 & \text{if there exists } i \in [n], x[i] = y[i] = 1 \\ 1 & \text{otherwise} \end{cases}$$

It is proved that the trivial protocol for DISJ_n, where Alice sends her input x to Bob, is the best we can do for this problem.

Proposition 4.1 ([Kushilevitz 1997])**.** $D(\mathsf{DISJ}_n) \geq n$

We study a variant of the DISJ_n function, called the GRAPHIC DISJOINTNESS function. Let G be a graph on n vertices and m edges. Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be two hereditary graph families. The following function is defined for the graph G, and the families $\mathcal{F}_1, \mathcal{F}_2$ as follows. For any two vertex subsets V_1 and V_2 such that $G[V_1] \in \mathcal{F}_1$ and $G[V_2] \in \mathcal{F}_2$,

$$\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}(V_1,V_2) = \begin{cases} 1 & \text{if } V_1 \cap V_2 = \emptyset \\ 0 & \text{otherwise} \end{cases}$$

A problem, related to that of computing $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$, is the problem of enumerating separating families for two graph families.

Definition 4.3. *Let G be a graph on n vertices, $\mathcal{F}_1$ and $\mathcal{F}_2$ be two graph families. Suppose $\mathcal{F}$ is a family of subsets of $V(G)$ with the following property: If we take any two vertex disjoint induced subgraphs $G_1, G_2 \leq_s G$, such that $G_1 \in \mathcal{F}_1$ and $G_2 \in \mathcal{F}_2$, there is a set $F \in \mathcal{F}$ such that $V(G_1) \subseteq F$ and $V(G_2) \cap F = \emptyset$. Then $\mathcal{F}$ is called an $(\mathcal{F}_1, \mathcal{F}_2)$-separating family in G. Such a set F is called a separating set for G_1 and G_2.*

We have the following Observation about a separating set.

Observation 4.1. *Let G be an n-vertex graph. Let G_1, G_2 be induced subgraphs of G. Suppose for each vertex $v \in V(G_1)$, $\overline{d}_G(v) < n/2$. Also, for each vertex $w \in V(G_2)$, $d_G(v) < n/2$. Then, $V(G_1) \cap V(G_2) = \emptyset$ and $S = \{v \mid v \in V(G), \overline{d}_G(v) < n/2\}$ is a separating set for G_1 and G_2.*

Proof. By definition, any vertex $v \in V(G_1)$ has $\overline{d}_G(v) < n/2$. Next, we show that for any vertex $w \in V(G_2)$, $\overline{d}_G(w) \geq n/2$. In the graph, any vertex v satisfies the condition $d_G(v) + \overline{d}_G(v) = n - 1$. For a vertex w with $d_G(v) < n/2$, $n/2 + \overline{d}_G(w) > n - 1$. In other words, $\overline{d}_G(w) > n/2 - 1$. This implies that $\overline{d}_G(w) \geq n/2$. Thus, any vertex v of $V(G_1)$ has $\overline{d}_G(v) < n/2$, while any vertex w of $V(G_2)$ has $\overline{d}_G(w) \geq n/2$. Then, $V(G_1) \cap V(G_2) = \emptyset$. By definition of S, every vertex of $V(G_1)$ belongs to S while no vertex of $V(G_2)$ belongs to S. By definition of a separating set, S is a separating set for G_1 and G_2. □

4.3 Communication protocols for pairs of Hereditary graph families

One of our main results is Theorem 4.1. To prove Theorem 4.1, we first need to prove a sub-linear communication complexity bound for a specific pair of graph families. More formally, in this section we consider a pair of hereditary families of graphs, $\mathcal{C}_r = \{H \mid \alpha(H) \leq r\}$ and $\mathcal{I}_\ell = \{H \mid \omega(H) \leq \ell\}$. Here, r and ℓ are two positive integers. In this section, we consider the communication complexity of $\mathsf{GDISJ}_{G,\mathcal{C}_r,\mathcal{I}_\ell}$. Using this, we complete the proof of Theorem 4.1. In the later half of this Section, we give upper bounds on the communication complexity of the function $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$, in terms of a structural parameter of the graph G. We consider two such structural parameters and design protocols with the help of this additional parameter.

4.3.1 Communication Protocol for Families of Sparse and Dense graphs

We will describe a communication protocol for $\mathsf{GDISJ}_{G,\mathcal{C}_r,\mathcal{I}_\ell}$, with complexity $o(n)$, for any positive constants r, ℓ. Here, Alice receives an induced subgraph G_1 of G such that $G_1 \in \mathcal{C}_r$, and Bob receives an induced subgraph G_2 of G such that $G_2 \in \mathcal{I}_\ell$. They have to determine whether the vertex sets of these two subgraphs of G are disjoint or not. Note that both Alice and Bob receive the graph G. The following simple observation is useful for making a protocol for $\mathsf{GDISJ}_{G,\mathcal{C}_r,\mathcal{I}_\ell}$.

Observation 4.2. *Let G_1, G_2 be two graphs such that $\alpha(G_1) \leq r$ and $\omega(G_2) \leq \ell$. Then,*

1. *for any vertex $v \in V(G_1)$, $\overline{N}_{G_1}(v)$ induces a subgraph G_1' of G_1 where $\alpha(G_1') \leq r-1$, and*

2. *for any vertex $v \in V(G_2)$, $N_{G_2}(v)$ induces a subgraph G_2' of G_2 where $\omega(G_2') \leq \ell-1$.*

First, we give a protocol $\Pi_{1,2}$ for $\mathsf{GDISJ}_{G,\mathcal{C}_1,\mathcal{I}_2}$, with a cost of $\mathcal{O}(\log^3 n)$. This protocol uses a protocol $\Pi_{1,1}$, for $\mathsf{GDISJ}_{G,\mathcal{C}_1,\mathcal{I}_1}$, as a sub-protocol. As mentioned earlier, for any pair of induced subgraphs $C, I \in G$, with $C \in \mathcal{C}_1, I \in \mathcal{I}_1$, $\mathsf{GDISJ}_{G,\mathcal{C}_1,\mathcal{I}_1}(C, I) = 1$ if and only if $\mathsf{CIS}_G(C, I) = 0$. The function CIS_G has a deterministic protocol of cost $\mathcal{O}(\log^2 n)$ [Yannakakis 1991]. Therefore, there is a protocol $\Pi_{1,1}$ of cost $\mathcal{O}(\log^2 n)$ for $\mathsf{GDISJ}_{G,\mathcal{C}_1,\mathcal{I}_1}$. The protocol for the general case $\mathsf{GDISJ}_{G,\mathcal{C}_r,\mathcal{I}_\ell}$ can be designed in a recursive manner that uses protocols of $\mathsf{GDISJ}_{G,\mathcal{C}_r,\mathcal{I}_{\ell-1}}$ and $\mathsf{GDISJ}_{G,\mathcal{C}_{r-1},\mathcal{I}_\ell}$ as subprotocols.

Lemma 4.1. *$D(\mathsf{GDISJ}_{G,\mathcal{C}_1,\mathcal{I}_2}) = \mathcal{O}(\log^3 n)$, where $n = |V(G)|$.*

Proof. Let Alice get the induced subgraph G_1 and Bob get the induced subgraph G_2. The following is a protocol $\Pi_{1,2}$ that Alice and Bob will execute. Alice and Bob continue with the protocol till either they detect a vertex in the intersection of $V(G_1)$ and $V(G_2)$, or one of G, G_1 and G_2 becomes an empty graph. The protocol is executed in top down fashion, i.e., the two players resort to a step only if the previous steps are not applicable.

1. If either G_1 or G_2 is an empty graph, then the players declare that the graphs are disjoint and stop.

2. Alice looks for a vertex $v \in V(G_1)$ with $\overline{d}_G(v) \geq n/2$. She sends the vertex v to Bob. If $v \in V(G_2)$, then Bob lets Alice know and they terminate the protocol. Otherwise, both players delete the vertices of $\overline{N}_G(v) \cup \{v\}$ from the graph G to obtain graph $G' = G - (\overline{N}_G(v) \cup \{v\})$. Alice defines $G'_1 = G_1 - \{v\}$ while Bob defines $G'_2 = G_2 - \overline{N}_G(v)$. Then, they continue the protocol for determining whether $V(G'_1) \cap V(G'_2) = \emptyset$ in G'.

3. Suppose Alice cannot find such a vertex v. Then Bob looks for a vertex $v \in V(G_2)$ with $d_G(v) \geq n/2$. Bob sends the vertex v to Alice. If $v \in V(G_1)$, then Alice lets Bob know and they terminate the protocol. Otherwise, both players use the protocol $\Pi_{1,1}$ to compute $\mathsf{GDISJ}_{G[N_G(v)],\mathcal{C}_1,\mathcal{I}_1}(G[N_G(v) \cap V(G_1)], G[N_G(v) \cap V(G_2)])$. If the output is 0, then they declare that $V(G_1) \cap V(G_2) \neq \emptyset$ and stop. Otherwise, they delete the vertices of $N_G[v]$ from G to get $G' = G - N_G[v]$. Alice defines $G'_1 = G_1 - N_G[v]$ while Bob defines $G'_2 = G_2 - N_G[v]$. Then, they continue the protocol for determining whether $V(G'_1) \cap V(G'_2) = \emptyset$ in G'.

4. Suppose all the above steps fail, then, both players declare that $V(G_1) \cap V(G_2) = \emptyset$ and stop.

First, we show the correctness of the protocol by induction on the size of $V(G)$. In the base case, G is an empty graph, and the protocol correctly returns that there is no intersection between $V(G_1)$ and $V(G_2)$. Suppose that the protocol correctly computes $\mathsf{GDISJ}_{G',\mathcal{C}_1,\mathcal{I}_2}$, for all G' such that $|V(G')| < n$. Now, consider a graph G on n vertices. Let $G_1 \in \mathcal{C}_1$ and $G_2 \in \mathcal{I}_2$ be the respective subgraphs of G given to Alice and Bob. If one of G_1 or G_2 is an empty graph, then clearly $V(G_1) \cap V(G_2) = \emptyset$ and the players correctly declare it in step 1.

1. Suppose there is a vertex $v \in V(G_1)$ with $\overline{d}_G(v) \geq n/2$. According to the protocol, Alice will send the vertex v to Bob. If $v \in V(G_2)$, then Bob lets Alice know and they terminate the protocol. Otherwise, both players know that $V(G_1) \subseteq N[v]$. Thus, it is enough to check whether there is a vertex w that belongs to both $V(G_1) \setminus \{v\}$ and $V(G_2) \setminus \overline{N}_G(v)$. This is equivalent to deleting the vertices $\overline{N}_G(v) \cup \{v\}$ from the graph G and running the protocol for the new graph G', that has strictly less number of vertices. In this protocol, Alice gets the graph $G'_1 = G_1 - \{v\}$ and Bob gets the graph $G'_2 = G_2 - \overline{N}_G(v)$. Since $\mathcal{C}_1$ and $\mathcal{I}_2$ are hereditary families, $G'_1 \in \mathcal{C}_1$ and $G'_2 \in \mathcal{I}_2$. By induction hypothesis, the protocol correctly computes $\mathsf{GDISJ}_{G',\mathcal{C}_1,\mathcal{I}_2}$. This implies that the protocol computes $\mathsf{GDISJ}_{G,\mathcal{C}_1,\mathcal{I}_2}$ in this case.

2. Suppose G_1 does not have any vertex with non-degree at least $n/2$. Let there be a vertex $v \in V(G_2)$ with $d_G(v) \geq n/2$. Bob sends the vertex v to Alice. If $v \in V(G_1)$, then Alice lets Bob know and they terminate the protocol. Otherwise, by Observation 4.2, both players know that $\omega(G[N_G(v)] \cap G_2) \leq 1$. It is also true that $\alpha(G[N_G(v)] \cap G_1) \leq 1$. Thus, Alice and Bob use the protocol $\Pi_{1,1}$ to solve the problem $\mathsf{GDISJ}_{G[N_G(v)],\mathcal{C}_1,\mathcal{I}_1}(G[N_G(v) \cap V(G_1)], G[N_G(v) \cap V(G_2)])$. If the output is 0, then they know that $(N_G(v) \cap V(G_1)) \cap (N_G(v) \cap V(G_2)) \neq \emptyset$, which also implies that $V(G_1) \cap V(G_2) \neq \emptyset$. Otherwise, since v is also not in the intersection of $V(G_1) \cap V(G_2)$, the players only need to determine an intersection in $G_1 - N_G[v]$ and $G_2 - N_G[v]$. This is equivalent to deleting the vertices of $N_G[v]$ from G to obtain the

graph G', which has strictly less number of vertices, and running the protocol for G'. In this protocol, Alice gets the graph $G'_1 = G_1 - N_G[v]$ and Bob gets the graph $G'_2 = G_2 - N_G[v]$. Since $\mathcal{C}_1$ and $\mathcal{I}_2$ are hereditary families, $G'_1 \in \mathcal{C}_1$ and $G'_2 \in \mathcal{I}_2$. By induction hypothesis, the protocol correctly computes $\mathsf{GDISJ}_{G',\mathcal{C}_1,\mathcal{I}_2}$. This implies that the protocol computes $\mathsf{GDISJ}_{G,\mathcal{C}_1,\mathcal{I}_2}$ in this case.

3. Suppose the above two cases do not hold. Then every vertex $v \in V(G_1)$ has $\overline{d}_G(v) < n/2$, while every vertex $w \in V(G_2)$ has $d_G(w) < n/2$. Therefore, by Observation 4.1, $V(G_1) \cap V(G_2) = \emptyset$ and the protocol gives the correct answer in step 4.

This proves the correctness of the protocol. At each round of the protocol, in the worst case, $\Pi_{1,1}$ is used as a sub-protocol. Since in every round we reduce the vertex set of G by at least half, there can be at most $\log n$ rounds before G becomes an empty graph. Thus, the cost of the protocol is $\mathcal{O}(\log^3 n)$. This completes the proof. □

Corollary 4.1. *$D(\mathsf{GDISJ}_{G,\mathcal{C}_2,\mathcal{I}_1}) = \mathcal{O}(\log^3 n)$, where $n = |V(G)|$.*

Proof. Suppose Alice gets the induced subgraph G_1 and Bob gets the induced subgraph G_2. Notice that in the graph $\overline{G}$, the induced subgraph $\overline{G_2}$ belongs to $\mathcal{C}_2$. Also, the induced subgraph $\overline{G_1}$ belongs to $\mathcal{I}_1$. Thus, the protocol $\Pi_{1,2}$, that computes $\mathsf{GDISJ}_{\overline{G},\mathcal{C}_2,\mathcal{I}_1}(\overline{G_2}, \overline{G_1})$, can also be used to compute $\mathsf{GDISJ}_{G,\mathcal{C}_2,\mathcal{I}_1}(G_1, G_2)$. □

We can give a protocol $\Pi_{r,\ell}$, for the problem $\mathsf{GDISJ}_{G,\mathcal{C}_r,\mathcal{I}_\ell}$, of cost $\mathcal{O}(\log^{r+\ell} n)$, using a protocol for $\Pi_{r,\ell-1}$ or $\Pi_{r-1,\ell}$. We use similar arguments as in the protocol $\Pi_{1,2}$, to design the protocol $\Pi_{r,\ell}$. Thus, we can prove the following theorem using induction on $r + \ell$.

Lemma 4.2. *For any $r, \ell \in \mathbb{N}$, $D(\mathsf{GDISJ}_{G,\mathcal{C}_r,\mathcal{I}_\ell}) = \mathcal{O}(\log^{r+\ell} n)$, where $n = |V(G)|$.*

Proof. Notice that $r, \ell > 0$. We prove the theorem by induction on $r + \ell$. When $r + \ell = 2$, it must be the case that $r = \ell = 1$ and the protocol $\Pi_{1,1}$ has complexity $\mathcal{O}(\log^2 n)$. When $r + \ell = 3$, then either $r = 1$ and $\ell = 2$, or $r = 2$ and $\ell = 1$. In this case Lemma 4.1 and Corollary 4.1 give protocols that have complexity $\mathcal{O}(\log^3 n)$. Thus, in the base cases, this hypothesis holds.

Suppose, the hypothesis is true for all pairs (r', ℓ') when $r' + \ell' < m$, for a positive integer m. Let (r, ℓ) be a pair of positive integers such that $r + \ell = m$. We design a protocol $\Pi_{r,\ell}$ for computing $\mathsf{GDISJ}_{G,\mathcal{C}_r,\mathcal{I}_\ell}$. Let Alice get an induced subgraph G_1 of G, and Bob get an induced subgraph G_2 of G. The protocol is as follows. Alice and Bob continue with the protocol till either they detect a vertex in the intersection of $V(G_1)$ and $V(G_2)$, or one of G, G_1 and G_2 becomes an empty graph. The protocol is executed in top down fashion, i.e., the two players resort to a step only if the previous steps are not applicable.

1. If either G_1 or G_2 is an empty graph, then the players declare that the graphs are disjoint and stop.

2. Alice looks for a vertex $v \in V(G_1)$ with $\overline{d}_G(v) \geq n/2$. Alice sends the vertex v to Bob. If $v \in V(G_2)$, then Bob lets Alice know and they terminate the protocol. Otherwise, both players use the protocol $\Pi_{r-1,\ell}$ to solve the problem $\mathsf{GDISJ}_{G[N_G(v)],\mathcal{C}_{r-1},\mathcal{I}_\ell}(G[\overline{N}_G(v) \cap V(G_1)], G[\overline{N}_G(v) \cap V(G_2)])$. If the output is 0, then they declare that $V(G_1) \cap V(G_2) \neq \emptyset$. Otherwise, they delete the vertices of $\overline{N}_G(v) \cup \{v\}$ from G to obtain graph $G' = G - (\overline{N}_G(v) \cup \{v\})$. Alice defines $G'_1 = G_1 - \{v\}$ while Bob defines $G'_2 = G_2 - \overline{N}_G(v)$. Then, they continue with the protocol for determining whether $V(G'_1) \cap V(G'_2) = \emptyset$ in G'.

3. Suppose Alice cannot find such a vertex v. Then Bob looks for a vertex $v \in V(G_2)$ with $d_G(v) \geq n/2$. Bob sends the vertex v to Alice. If $v \in V(G_1)$, then Alice lets Bob know and they terminate the protocol. Otherwise, both players use the protocol $\Pi_{r,\ell-1}$ to solve the problem $\mathsf{GDISJ}_{G[N_G(v)],\mathcal{C}_r,\mathcal{I}_{\ell-1}}(G[N_G(v) \cap V(G_1)], G[N_G(v) \cap V(G_2)])$. If the output is 0, then they declare that $V(G_1) \cap V(G_2) \neq \emptyset$. Otherwise, they delete the vertices of $N_G[v]$ from G to get $G' = G - N_G[v]$. Alice defines $G'_1 = G_1 - N_G[v]$ while Bob defines $G'_2 = G_2 - N_G[v]$. Then, they continue with the protocol for determining whether $V(G'_1) \cap V(G'_2) = \emptyset$ in G'.

4. Suppose all the above cases do not hold, then, both players declare that there can be no intersection between $V(G_1)$ and $V(G_2)$.

We argue the correctness of the protocol, under the induction hypothesis that the desired protocols exist corresponding to all pairs (r', ℓ') when $r' + \ell' < m$. We show the correctness of the protocol by induction on the size of $V(G)$. In the base case, G is an empty graph, and the protocol correctly returns that there is no intersection between $V(G_1)$ and $V(G_2)$. Now, suppose that the protocol correctly computes $\mathsf{GDISJ}_{G',\mathcal{C}_r,\mathcal{I}_\ell}$, for all G' such that $|V(G')| < n$. Now, consider a graph G on n vertices. Let $G_1 \in \mathcal{C}_1$ and $G_2 \in \mathcal{I}_2$ be the respective subgraphs of G given to Alice and Bob repectively.

If one of G_1 or G_2 is an empty graph, then clearly $V(G_1) \cap V(G_2) = \emptyset$ and they correctly declare it in step 1.

1. Suppose there is a vertex $v \in V(G_1)$ with $\overline{d}_G(v) \geq n/2$. Alice sends the vertex v to Bob. If $v \in V(G_2)$, then Bob lets Alice know and they terminate the protocol. Otherwise, by Observation 4.2, both players know that $\alpha(G[\overline{N}_G(v)] \cap G_1) \leq r - 1$ while $\omega(G[\overline{N}_G(v)] \cap G_2) \leq \ell$. Thus, Alice and Bob use the protocol $\Pi_{r-1,\ell}$ to solve the problem $\mathsf{GDISJ}_{G[\overline{N}_G(v)],\mathcal{C}_{r-1},\mathcal{I}_\ell}(G[\overline{N}_G(v) \cap V(G_1)], G[\overline{N}_G(v) \cap V(G_2)])$. If the output is 0, then they know that $(\overline{N}_G(v) \cap V(G_1)) \cap (\overline{N}_G(v) \cap V(G_2)) \neq \emptyset$, which also implies that $V(G_1) \cap V(G_2) \neq \emptyset$. Otherwise, they only need to determine a vertex intersection in $G_1 - (\overline{N}_G(v) \cup \{v\})$ and $G_2 - \overline{N}_G(v)$. This is equivalent to deleting the vertices of $\overline{N}_G(v) \cup \{v\}$ from G, thereby obtaining a graph G' where the size of the vertex set is at most half of that of G. The players run the protocol on G'. In this protocol, Alice gets the graph $G'_1 = G_1 - (\overline{N}_G(v) \cup \{v\})$ and Bob gets the graph $G'_2 = G_2 - (\overline{N}_G(v) \cup \{v\})$. By definition of the families $\mathcal{C}_r$ and $\mathcal{I}_\ell$, $G'_1 \in \mathcal{C}_r$ and $G'_2 \in \mathcal{I}_\ell$. By induction hypothesis, the protocol correctly computes $\mathsf{GDISJ}_{G',\mathcal{C}_r,\mathcal{I}_\ell}$. This implies that the protocol computes $\mathsf{GDISJ}_{G,\mathcal{C}_r,\mathcal{I}_\ell}$ in this case.

2. Suppose G_1 does not have a vertex with non-degree at least $n/2$. Let $v \in V(G_2)$ have

$d_G(v) \geq n/2$. Bob sends the vertex v to Alice. If $v \in V(G_1)$, then Alice lets Bob know and they terminate the protocol. Otherwise, by Observation 4.2, both players know that $\omega(G[N_G(v) \cap V(G_2)]) \leq \ell - 1$ while $\alpha(G[N_G(v) \cap V(G_1)]) \leq r$. Thus, Alice and Bob use the protocol $\Pi_{r,\ell-1}$ to solve the problem $\mathsf{GDISJ}_{N_G(v),\mathcal{C}_r,\mathcal{I}_{\ell-1}}(G[N_G(v) \cap V(G_1)], G[N_G(v) \cap V(G_2)])$. If the output is 0, then they know that $(N_G(v) \cap V(G_1)) \cap (N_G(v) \cap V(G_2)) \neq \emptyset$, which also implies that $V(G_1) \cap V(G_2) \neq \emptyset$. Otherwise, they only need to determine a vertex intersection in $G_1 - N_G[v]$ and $G_2 - N_G[v]$. This is equivalent to deleting the vertices of $N_G[v]$ from G, thereby obtaining a graph G' where the size of the vertex set is at most half of that of G. The players run the protocol on the new graph G'. In this protocol, Alice gets the graph $G'_1 = G_1 - N_G[v]$ and Bob gets the graph $G'_2 = G_2 - N_G[v]$. By definition of the families $\mathcal{C}_r$ and $\mathcal{I}_\ell$, $G'_1 \in \mathcal{C}_r$ and $G'_2 \in \mathcal{I}_\ell$. By induction hypothesis, the protocol correctly computes $\mathsf{GDISJ}_{G',\mathcal{C}_r,\mathcal{I}_\ell}$. This implies that the protocol computes $\mathsf{GDISJ}_{G,\mathcal{C}_r,\mathcal{I}_\ell}$ in this case.

3. Suppose the above cases do not hold. Then every vertex v of $V(G_1)$ has $\overline{d}_G(v) < n/2$, while every vertex $w \in V(G_2)$ has $d_G(w) < n/2$. Therefore, by Observation 4.1, there can be no intersection between $V(G_1)$ and $V(G_2)$ and the protocol gives the correct answer due to step 4.

This proves the correctness of the protocol. At each round of the protocol, in the worst case, $\Pi_{r-1,\ell}$ or $\Pi_{r,\ell-1}$ is used as a sub-protocol. Since in every round we reduce the vertex set of G by half, there can be at most $\log n$ rounds before G becomes an empty graph. By induction hypothesis, both $\Pi_{r-1,\ell}$ and $\Pi_{r,\ell-1}$ have complexity $\mathcal{O}(\log^{r+\ell-1} n)$. It follows that the complexity of the protocol $\Pi_{r,\ell}$ is $\mathcal{O}(\log^{r+\ell} n)$. □

4.3.2 Characterization for Hereditary graph families

We are ready to prove Theorem 4.1. That is, we try to determine $D(\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2})$ for any given pair of hereditary families $\mathcal{F}_1, \mathcal{F}_2$. If one of $\mathcal{F}_1$ or $\mathcal{F}_2$ is finite, then the number of vertices of each graph in one of the families is bounded by a constant. Thus, a trivial protocol would be for the player, who receives the bounded-sized subgraph, to send the full subgraph over to the other player, using $\mathcal{O}(\log n)$ bits. This implies, $D(\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}) = \mathcal{O}(\log n)$. So, the interesting case is when both $\mathcal{F}_1$ and $\mathcal{F}_2$ are infinite. Now we prove Theorem 4.1.

Theorem 4.1. *For any two hereditary families of graphs $\mathcal{F}_1$ and $\mathcal{F}_2$, for any integer $n > 0$, there is an n-vertex graph G such that $D(\mathsf{GDIS}_{G,\mathcal{F}_1,\mathcal{F}_2}) = \Omega(n)$ if and only if $\mathcal{F}_1 \cap \mathcal{F}_2$ is an infinite family.*

Proof. Without loss of generality we can assume that both $\mathcal{F}_1$ and $\mathcal{F}_2$ are infinite. Suppose the intersection family is finite. This means that there is a constant r such that a complete graph K_r, on r vertices, does not belong to the intersection family, because of finiteness. Since K_r does not belong to $\mathcal{F}_1 \cap \mathcal{F}_2$, it must not belong to at least one of the families. Let this be $\mathcal{F}_1$. Since $\mathcal{F}_1$ is hereditary, no graph in $\mathcal{F}_1$ has K_r as an induced subgraph. This implies that for any graph G in $\mathcal{F}_1$, $\omega(G) \leq r - 1$. Now we show that for any graph G in $\mathcal{F}_2$, $\alpha(G) \leq \ell - 1$ for some constant ℓ. Towards that, we first claim that $\mathcal{F}_1$ contains all stable graphs. Otherwise, since $\mathcal{F}_1$ is a hereditary family, if $\mathcal{F}_1$ does not contain a stable graph on ℓ' vertices, all graphs in $\mathcal{F}_1$ neither have a r-sized clique as an induced subgraph nor an ℓ'-sized independent set as an induced subgraph. However, by Ramsey's theorem,

each graph in $\mathcal{F}_1$ has at most $R(r, \ell')$ vertices, thus contradicting the infiniteness of $\mathcal{F}_1$. So far we know that, $\mathcal{F}_1 \cap \mathcal{F}_2$ is finite and $\mathcal{F}_1$ contains all stable graphs. This implies that $\mathcal{F}_2$ does not contain all stable graphs. Let ℓ be an integer such that $\overline{K}_\ell \notin \mathcal{F}_2$. By the hereditary property of $\mathcal{F}_2$, no graph in $\mathcal{F}_2$ contains $\overline{K}_\ell$ as an induced subgraph. That is, for all graph G in $\mathcal{F}_2$, $\alpha(G) \leq \ell - 1$. Thus, Lemma 4.2 gives us a deterministic protocol for $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$, with $o(n)$ communication complexity.

For the reverse direction, suppose the intersection family $\mathcal{F}_1 \cap \mathcal{F}_2$ is an infinite family. To prove $D(\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}) = \Omega(n)$ we give a simple reduction from DISJ_n. As $\mathcal{F}_1 \cap \mathcal{F}_2$ has infinitely many graphs, there are infinitely many integers $n > 0$ such that an n-vertex graph belongs to $\mathcal{F}_1 \cap \mathcal{F}_2$. Consider any integer $n_0 > 0$. There must be an integer $n > n_0$ such that an n-vertex graph G belongs to $\mathcal{F}_1 \cap \mathcal{F}_2$. Since $\mathcal{F}_1$ and $\mathcal{F}_2$ are hereditary families, $\mathcal{F}_1 \cap \mathcal{F}_2$ is also a hereditary graph family. Hence, any n_0-vertex induced subgraph of G belongs to $\mathcal{F}_1 \cap \mathcal{F}_2$. To summarise, we know that (i) for any integer $n > 0$, there is an n-vertex graph G in the intersection family $\mathcal{F}_1 \cap \mathcal{F}_2$ and (ii) $\mathcal{F}_1 \cap \mathcal{F}_2$ is a hereditary family. Therefore, for such a graph G, any subgraph H is in the family $\mathcal{F}_1$ as well as in the family $\mathcal{F}_2$.

Now our reduction works as follows. In DISJ_n, Alice is given $x \in \{0,1\}^n$ and Bob is given $y \in \{0,1\}^n$ and they want to check whether there is an $i \in [n]$ such that $x[i] = y[i] = 1$. Now we create an instance of $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$ as follows. We fix an n-vertex graph $G \in \mathcal{F}_1 \cap \mathcal{F}_2$, with $V(G) = \{v_1, \ldots, v_n\}$. Let $V_x = \{v_i \in V(G) \mid i \in [n], x[i] = 1\}$ and $V_y = \{v_i \in V(G) \mid i \in [n], y[i] = 1\}$. Since $G \in \mathcal{F}_1 \cap \mathcal{F}_2$, $G[V_x] \in \mathcal{F}_1$ and $G[V_y] \in \mathcal{F}_2$. In the $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$ problem, Alice gets $G[V_x]$ and Bob gets $G[V_y]$. Clearly $V_x \cap V_y \neq \emptyset$ if and only if there is an $i \in [n]$ such that $x[i] = y[i] = 1$. Hence, by Proposition 4.1, $D(\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}) = \Omega(n)$. This concludes the proof. □

For the rest of this chapter, a pair $(\mathcal{F}_1, \mathcal{F}_2)$ of hereditary graph families where $\mathcal{F}_1 \cap \mathcal{F}_2$ is a finite graph family, will be referred to as a *good pair of graph families*. The proof of Theorem 4.1 also gives us the following folklore corollary.

Corollary 4.2. *Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be a good pair of graph families. Then, there are constants r and ℓ, such that for any graph $G_1 \in \mathcal{F}_1$ and $G_2 \in \mathcal{F}_2$, $\omega(G_1) \leq r$ and $\alpha(G_2) \leq \ell$.*

Corollary 4.2 and Ramsey theorem leads us to another useful corollary.

Corollary 4.3. *Let G be a graph. Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be a good pair of graph families. Then there are constants r and ℓ (same as the constants mentioned in Corollary 4.2) such that, for any pair (G_1, G_2) of induced subgraphs of G, if $G_1 \in \mathcal{F}_1$ and $G_2 \in \mathcal{F}_2$, then $|V(G_1) \cap V(G_2)| < R(r+1, \ell+1)$.*

Proof. From Corollary 4.2, there are constants r, ℓ, such that for any graph $G_1 \in \mathcal{F}_1$ and $G_2 \in \mathcal{F}_2$, $\omega(G_1) \leq r$ and $\alpha(G_2) \leq \ell$. Then, the subgraph $G_1 \cap G_2$ is a graph where $\omega(G[V(G_1) \cap V(G_2)]) \leq r$ and $\alpha(G[V(G_1) \cap V(G_2)]) \leq \ell$. This implies that $|V(G_1) \cap V(G_2)| < R(r+1, \ell+1)$, where $R(\cdot)$ is the Ramsey function. Thus, the required property holds for this good pair of families. □

4.3.3 A Parameterized approach to designing protocols

In Section 4.3.1, for each pair of constants r, ℓ, we saw a protocol for $\mathsf{GDISJ}_{G,\mathcal{C}_r,\mathcal{I}_\ell}$ with sublinear communication complexity. We also showed that any good pair $(\mathcal{F}_1, \mathcal{F}_2)$ of graph families must be such that there are constants r, ℓ with $\mathcal{F}_1 \subseteq \mathcal{C}_r, \mathcal{F}_2 \subseteq \mathcal{I}_\ell$. In this section, we give an alternate protocol that uses the structure of the input graph G, to obtain a more refined upper bound on the communication complexity of $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$. For an n-vertex graph, let $\mathsf{opt}^G_{\mathcal{F}_1,\mathcal{F}_2}$ denote the size of a minimum set S of vertices such that $V(G) \setminus S$ has a bipartition (V_1, V_2) with $G[V_1] \in \mathcal{F}_1$ and $G[V_2] \in \mathcal{F}_2$. In this section, for an n-vertex graph G and a good pair of graph families $(\mathcal{F}_1, \mathcal{F}_2)$, we give a protocol for $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$ that has $\mathcal{O}(\log^c(\mathsf{opt}^G_{\mathcal{F}_1,\mathcal{F}_2}) + \log n)$ communication complexity. Here, c is a constant depending only on the two graph families.

Theorem 4.2. *Let G be an n-vertex graph and $(\mathcal{F}_1, \mathcal{F}_2)$ be a good pair of graph families. Also, let r and ℓ be the constants such that $\mathcal{F}_1 \subseteq \mathcal{C}_r$ and $\mathcal{F}_2 \subseteq \mathcal{I}_\ell$. Let $\mathsf{opt}^G_{\mathcal{F}_1,\mathcal{F}_2}$ be the size of a minimum set S of vertices such that $V(G) \setminus S$ has a bipartition (V_1, V_2) with $G[V_1] \in \mathcal{F}_1$ and $G[V_2] \in \mathcal{F}_2$. Then there is a protocol for $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$ that has $\mathcal{O}(\log^{r+\ell}(\mathsf{opt}^G_{\mathcal{F}_1,\mathcal{F}_2}) + \log n)$ communication complexity.*

Proof. We can assume that Alice and Bob both have an optimum vertex set S such that $V(G) \setminus S$ has a bipartition (V_1, V_2) with $G[V_1] \in \mathcal{F}_1$ and $G[V_2] \in \mathcal{F}_2$. Thus $|S| = \mathsf{opt}^G_{\mathcal{F}_1,\mathcal{F}_2}$. The players also have a bipartition (V_1, V_2) of $V(G) \setminus S$, such that $G[V_1] \in \mathcal{F}_1$ and $G[V_2] \in \mathcal{F}_2$.

Now let Alice receive the induced subgraph $G_1 \in \mathcal{F}_1$ and Bob receive the induced subgraph $G_2 \in \mathcal{F}_2$. The following is a protocol Π that Alice and Bob will execute. The protocol is executed in top down fashion, i.e., the two players resort to a step only if the previous steps are not applicable.

1. If either G_1 or G_2 is an empty graph, then they declare that the graphs are disjoint and stop.
2. Alice and Bob run the protocol $\Pi_{r,\ell}$ to compute $\mathsf{GDISJ}_{G[S],\mathcal{C}_r,\mathcal{I}_\ell}(G_1[S], G_2[S])$. If there is a vertex intersection between $G_1[S]$ and $G_2[S]$, then they declare that the graphs G_1 and G_2 intersect and stop the protocol.
3. Suppose there is no vertex intersection between $G_1[S]$ and $G_2[S]$. Alice sends the vertices of the subgraph $G_1 \cap G[V_2]$ to Bob. If Bob finds that $V(G_2) \cap V(G_1 \cap G[V_2]) \neq \emptyset$, then Bob lets Alice know and they terminate the protocol.
4. Suppose Bob does not find any vertex common to both $V(G_1 \cap G[V_2])$ and $V(G_2)$. Then Bob sends the vertices of the subgraph $G_2 \cap G[V_1]$ to Alice. If Alice finds that $V(G_1) \cap V(G_2 \cap G[V_2]) \neq \emptyset$, then Alice lets Bob know and they terminate the protocol. Otherwise, they declare that the two graphs G_1 and G_2 do not intersect on any vertex and stop.

First, we show the correctness of the protocol. If one of G_1 or G_2 is an empty graph, there can be no vertex intersection and this is detected in step 1. By the definition of hereditary graph families, $G_1[S] \in \mathcal{F}_1$ and $G_2[S] \in \mathcal{F}_2$. By definition of the two families, this also means that $G_1[S] \in \mathcal{C}_r$ and $G_2[S] \in \mathcal{I}_\ell$. If $V(G_1[S]) \cap V(G_2[S]) \neq \emptyset$, then the

subprotocol $\Pi_{r,\ell}$ correctly detects the intersection in step 2. Otherwise, $V(G_1)$ and $V(G_2)$ can intersect either in V_1 or in V_2. If they intersect in V_2, then the intersection is detected in step 3. Otherwise, the intersection is detected in step 4. If neither of the above cases happen, then $V(G_1) \cap V(G_2) = \emptyset$. This is detected at the end of step 4. Thus, the protocol is correct in computing $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$.

Next, we show the bound on the communication complexity. Running the subprotocol $\Pi_{r,\ell}$ to compute $\mathsf{GDISJ}_{G[S],\mathcal{C}_r,\mathcal{I}_\ell}(G_1[S], G_2[S])$ takes $\mathcal{O}(\log^{r+\ell} \mathsf{opt}^G_{\mathcal{F}_1,\mathcal{F}_2})$ bits. By definition, $G_1 \in \mathcal{F}_1$ and $G[V_2] \in \mathcal{F}_2$. Then, by Corollary 4.3, $|V(G_1) \cap V(G[V_2])| < R(r+1, \ell+1)$. Thus, in step 3, Alice sends at most $R(r+1, \ell+1) \log n$ bits to Bob. By a similar argument, in step 4, Bob sends at most $R(r+1, \ell+1) \log n$ bits to Alice. Therefore, the communication complexity of Π is $\mathcal{O}(\log^{r+\ell}(\mathsf{opt}^G_{\mathcal{F}_1,\mathcal{F}_2}) + \log n)$. □

In fact, as a corollary, this gives us a more nuanced bound on the communication complexity of the CIS_G function.

Corollary 4.4. *Let G be an n-vertex graph. Let $\mathsf{opt}^G_{\mathcal{C}_1,\mathcal{I}_1}$ be the size of a minimum set S of vertices such that $V(G) \setminus S$ has a bipartition (V_1, V_2) where $G[V_1]$ is a clique and $G[V_2]$ is an independent set. Then there is a protocol for CIS_G that has $\mathcal{O}(\log^2(\mathsf{opt}^G_{\mathcal{C}_1,\mathcal{I}_1}) + \log n)$ communication complexity.*

Suppose $(\mathcal{F}_1, \mathcal{F}_2)$ is a pair of hereditary graph families that are not good. By Theorem 4.1, for an n-vertex graph G, any protocol for $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$ must have communication complexity $\Omega(n)$. This gives us the following corollary for our new notion of communication protocols.

Corollary 4.5. *Let $(\mathcal{F}_1, \mathcal{F}_2)$ be a pair of hereditary graph families that have infinitely many graphs in their intersection. Then, for each integer $n > 0$, there is a graph G such that for any computable function f, there cannot be a protocol for $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$, that has communication complexity $f(\mathsf{opt}^G_{\mathcal{F}_1,\mathcal{F}_2}) + o(n)$.*

Proof. We give a proof by contradiction. Let Π be a protocol for $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$, on any n-vertex graph G, that has communication complexity $f(\mathsf{opt}^G_{\mathcal{F}_1,\mathcal{F}_2}) + o(n)$, for some computable function f.

Let $\mathcal{G} = \{G \mid G \in \mathcal{F}_1 \cap \mathcal{F}_2\}$. By the definition of hereditary graph families, every subgraph of G belongs to $\mathcal{G}$. This means that $V(G)$ can always be bipartitioned into $V_1 \uplus V_2$ such that $G[V_1] \in \mathcal{F}_1$ and $G[V_2] \in \mathcal{F}_2$. Then, for any graph $G \in \mathcal{G}$, $\mathsf{opt}^G_{\mathcal{F}_1,\mathcal{F}_2} = 0$. Since, by definition, $\mathcal{G}$ is an infinite graph family, there are infinitely many non-negative integers n, such that an n-vertex graph G belongs to $\mathcal{G}$. Consider any integer $n_0 > 0$. There must be an integer $n > n_0$ such that an n-vertex graph G belongs to $\mathcal{F}_1 \cap \mathcal{F}_2$. Since $\mathcal{F}_1$ and $\mathcal{F}_2$ are hereditary families, $\mathcal{F}_1 \cap \mathcal{F}_2$ is also a hereditary graph family. Hence, any n_0-vertex induced subgraph of G belongs to $\mathcal{F}_1 \cap \mathcal{F}_2$. Therefore, (i) for any integer $n > 0$, there is an n-vertex graph G in the intersection family $\mathcal{F}_1 \cap \mathcal{F}_2$ and (ii) $\mathcal{F}_1 \cap \mathcal{F}_2$ is a hereditary family. This means, for such a graph G, any subgraph H is in the family $\mathcal{F}_1$ as well as in the family $\mathcal{F}_2$. We prove that $D(\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}) = \Omega(n)$ by giving a simple reduction from DISJ_n. In DISJ_n, Alice is given $x \in \{0,1\}^n$ and Bob is given $y \in \{0,1\}^n$ and they want to check whether there is an $i \in [n]$ such that $x[i] = y[i] = 1$. Now we create an instance of $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$ as follows. We fix an n-vertex graph $G \in \mathcal{F}_1 \cap \mathcal{F}_2$, with $V(G) = \{v_1, \ldots, v_n\}$. Let $V_x = \{v_i \in V(G) \mid i \in [n], x[i] = 1\}$ and $V_y = \{v_i \in V(G) \mid i \in [n], y[i] = 1\}$.

Since $G \in \mathcal{F}_1 \cap \mathcal{F}_2$, $G[V_x] \in \mathcal{F}_1$ and $G[V_y] \in \mathcal{F}_2$. In the $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$ problem, Alice gets $G[V_x]$ and Bob gets $G[V_y]$. Clearly $V_x \cap V_y \neq \emptyset$ if and only if there is an $i \in [n]$ such that $x[i] = y[i] = 1$. Hence, by Proposition 4.1, $D(\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}) = \Omega(n)$. However, the protocol Π computes $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$ using $f(0) + o(n) = o(n)$ bits, which is a contradiction. Therefore, no such protocol Π exists. □

4.3.4 Parameterizing by degeneracy

In Section 4.3.1, we considered a pair of graph families $(\mathcal{C}_r, \mathcal{I}_\ell)$ and gave a communication protocol for $\mathsf{GDISJ}_{G,\mathcal{C}_r,\mathcal{I}_\ell}$ of cost $\mathcal{O}(\log^{r+\ell} n)$. In this section, we exhibit a special case of sparse and dense graphs, where there is a protocol, for the disjointness problem, with communication complexity $\mathcal{O}(\log^2 n)$. A *d-degenerate graph* is an undirected graph in which every induced subgraph has a vertex of degree at most d. We consider the pair of families $(\mathcal{C}_1, \mathcal{D}_\ell)$, where $\mathcal{D}_\ell$ is the set of all ℓ-degenerate graphs and $\mathcal{C}_1$ is the set of all complete graphs. Note that $\mathcal{D}_0$ is the family of independent sets. Hence, this is still a generalisation of the Clique vs Independent Set problem. The following two observations are useful for our protocol.

Observation 4.3. *Let H be a d-degenerate graph. Then, $\omega(H) \leq d$.*

Observation 4.4. *Let H be a d-degenerate n-vertex graph. For $i \in \{0, 1, \ldots, 2d\}$, let $V_i = \{v \in V(H) \mid d_H(v) = i\}$. Let $V_{\leq 4d} = \bigcup_{0 \leq i \leq 4d} V_i$. Let $V_{>4d} = \{v \in V(H) \mid d_H(v) \geq 4d + 1\}$. Then $|V_{\leq 4d}| \geq n/2$.*

Proof. It is well known that in a d-degenerate graph there are at most dn edges. By the Handshaking Lemma, $\Sigma_{v \in V_{\leq 4d}} d_G(v) + \Sigma_{v \in V_{>4d}} d_G(v) \leq 2dn$. Suppose, for the sake of contradiction, that $n/2 > |V_{\leq 4d}|$. This implies that $|V_{>4d}| \geq n/2$. We know that $\Sigma_{v \in V_{>4d}} d_G(v) \geq (4d+1)|V_{>4d}| \geq (4d+1)\frac{n}{2}$. This means that $\Sigma_{v \in V_{>4d}} d_G(v) \geq 2dn + \frac{1}{2}n > 2dn$. However, this is a contradiction to the fact that $\Sigma_{v \in V_{\leq 4d}} d_G(v) + \Sigma_{v \in V_{>4d}} d_G(v) \leq 2dn$. Hence, it must be the case that $|V_{\leq 4d}| \geq n/2$. □

Now we are ready to prove the following theorem regarding the communication complexity of $\mathsf{GDISJ}_{G,\mathcal{C}_1,\mathcal{D}_\ell}$.

Theorem 4.3. *For any constant $\ell \in \mathbb{N}$, $D(\mathsf{GDISJ}_{G,\mathcal{C}_1,\mathcal{D}_\ell}) = \mathcal{O}(\ell \log^2 n)$, where $n = |V(G)|$.*

Proof. We design a protocol for $\mathsf{GDISJ}_{G,\mathcal{C}_1,\mathcal{D}_\ell}$ with the required cost. Here G is a fixed graph known to both Alice and Bob. Alice gets an induced subgraph G_1 of G and Bob gets an induced subgraph G_2 of G such that G_1 is a clique and G_2 is ℓ-degenerate. The following is the protocol that Alice and Bob will execute. Alice and Bob continue with the protocol till either they detect a vertex in the intersection of $V(G_1)$ and $V(G_2)$, or one of G, G_1 and G_2 becomes an empty graph. The two players choose to execute a step only if the previous steps are not applicable.

1. If G_1 or G_2 is an empty graph, then they declare that $V(G_1) \cap V(G_2) = \emptyset$ and stop.

2. Alice looks for a vertex $v \in V(G_1)$ with $\overline{d}_G(v) \geq n/2$. Alice sends the vertex v to Bob. If $v \in V(G_2)$, then Bob lets Alice know and they terminate the protocol. Otherwise, both players delete the vertices of $\overline{N}_G(v) \cup \{v\}$ from the graph G to obtain graph $G' = G - (\overline{N}_G(v) \cup \{v\})$. Alice defines $G'_1 = G_1 - \{v\}$ while Bob defines $G'_2 = G_2 - \overline{N}_G(v)$. Then, they continue the protocol for determining whether $V(G'_1) \cap V(G'_2) = \emptyset$ in G'.

3. Suppose Alice cannot find such a vertex v. Then Bob looks for a vertex $v \in V(G_2)$ with $d_{G_2}(v) \leq 4\ell$ and $d_G(v) \geq n/2$. Bob sends $N_{G_2}[v]$ to Alice. If $N_{G_2}[v] \cap V(G_1) \neq \emptyset$, then Alice lets Bob know and they terminate the protocol. Otherwise, both players modify the graph G by deleting the vertices of $N_G[v]$. They also modify their respective graphs G_1 and G_2 by deleting the vertices of $N_G[v]$ from them. Then, they run the protocol with the new graphs.

4. Suppose neither player is successful. Let $A' = \{w \in V(G_2) \mid d_{G_2}(w) \leq 4\ell\}$. Alice and Bob run this protocol, as a subprotocol, to determine whether there is any vertex intersection for $G_1 - A'$ and G_2. If they find any vertex intersection, then they declare that there is a vertex intersection in $V(G_1)$ and $V(G_2)$. Otherwise, they declare that $V(G_1) \cap V(G_2) = \emptyset$ and stop.

We show the correctness of the protocol by induction on $|V(G)| + |V(G_2)|$. In the base case when $|V(G)| + |V(G_2)| = 0$, the graph G is an empty graph, and the protocol returns the correct answer in step 1. Suppose that the protocol correctly solves $\mathsf{GDISJ}_{G',\mathcal{C}_1,\mathcal{D}_\ell}$, for all G' such that $|V(G')| < n$. Now, consider a graph G on n vertices. Let $G_1 \in \mathcal{C}_1$ and $G_2 \in \mathcal{D}_\ell$ be the respective induced subgraphs of G given to Alice and Bob. If G_1 or G_2 is an empty graph, then the protocol gives the correct answer in step 1. Otherwise, we have the following cases.

1. Let there be a vertex $v \in V(G_1)$ with $\overline{d}_G(v) \geq n/2$. Alice sends the vertex v to Bob. If v is also contained in $V(G_2)$, then protocol gives the correct answer in step 2. Otherwise, it is true that $V(G_1) \cap \overline{N}_G(v) = \emptyset$. This also means that $(V(G_1) \cap V(G_2)) \cap (\overline{N}_G(v) \cup \{v\}) = \emptyset$. Thus, now it is enough to determine whether there is a vertex in $N_G(v)$ that belongs to both $V(G_1)$ and $V(G_2)$. This is equivalent to running the protocol on $G' = G - (\overline{N}_G(v) \cup \{v\})$ for the induced subgraphs $G'_1 = G_1 - (\overline{N}_G(v) \cup \{v\})$ and $G'_2 = G_2 - (\overline{N}_G(v) \cup \{v\})$. Notice that, since $\mathcal{C}_1$ and $\mathcal{D}_\ell$ are hereditary families, $G'_1 \in \mathcal{C}_1$ and $G'_2 \in \mathcal{D}_\ell$. Thus by induction hypothesis, the protocol gives correct answer in step 2.

2. Suppose the first case does not hold. Also, let there be a vertex $v \in V(G_2)$ with $d_{G_2}(v) \leq 4\ell$ and $d_G(v) \geq n/2$. Bob sends $N_{G_2}[v]$ to Alice. If $N_{G_2}[v] \cap V(G_1) \neq \emptyset$, then the protocol gives correct answer in step 3. Otherwise, $(V(G_1) \cap V(G_2)) \cap (N_G[v]) = \emptyset$. So, now it is enough to determine whether there is a vertex in $\overline{N}_G(v)$ that belongs to both $V(G_1)$ and $V(G_2)$. This is equivalent to running the protocol on $G' = G - N_G[v]$ for the induced subgraphs $G'_1 = G_1 - N_G[v]$ and $G'_2 = G_2 - N_G[v]$. Thus by induction hypothesis, the protocol gives correct answer in step 3.

3. Suppose the first two cases do not hold. Every vertex $v \in V(G_1)$ has $\overline{d}_G(v) < n/2$. Also, let $A' = \{w \in V(G_2) \mid d_{G_2}(w) \leq 4\ell\}$. Then every vertex $w \in A'$ has $d_G(w) < n/2$. By Observation 4.1, it is true that $V(G_1) \cap A' = \emptyset$. Hence, it is enough to

determine whether $V(G_1)$ and $V(G_2) \setminus A'$ intersect. We determine whether $V(G_1)$ and $V(G_2) \setminus A'$ intersect by running the protocol on G for the induced subgraphs G_1 and $G_2 - A'$. Since G_2 is an ℓ-degenerate graph, $A' \neq \emptyset$. Hence, by induction hypothesis, the protocol gives the correct answer in step 4.

Now we prove the cost of the protocol. In step 2 Alice sends $\log n$ bits and in step 3 Bob sends $4\ell \log n$ bits. In step 2 and 3, the cardinality of the new graph reduces by $1/2$ fraction of the original number of vertices in the graph. In step 4, by Observation 4.4, the number of vertices in $G_2 - A'$ is at most $\frac{1}{2}|V(G_2)|$. Thus after $\mathcal{O}(\log n + \log |G_2|)$ rounds, either the graph G becomes an empty graph or one of the induced subgraphs held by Alice of Bob becomes an empty graph. This implies that the cost of the protocol is $\mathcal{O}(\ell \log^2 n)$ □

4.4 Separating families

In this section, we design enumeration algorithms for separating families for a good pair of graph families. It was stated in [Lovász 1990] that a non-deterministic protocol for $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$ corresponds to a $(\mathcal{F}_1, \mathcal{F}_2)$-separating family. This means that if $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$ has non-deterministic, and hence deterministic, complexity c, then there exists a $(\mathcal{F}_1, \mathcal{F}_2)$-separating family of size 2^c. From Corollary 4.2 and Lemma 4.2, this means that, for an n-vertex graph, there exists a $(\mathcal{F}_1, \mathcal{F}_2)$-separating family of size $2^{\mathcal{O}(\log^{r+\ell} n)}$, for some constants r, ℓ. However, this does not mean that there is an enumeration algorithm that finds such a separating family in time $2^{\mathcal{O}(\log^{r+\ell} n)n^{\mathcal{O}(1)}}$. First, for an n-vertex graph G, we see an algorithm to enumerate a $(\mathcal{C}_r, \mathcal{I}_\ell)$-separating family of size $2^{\mathcal{O}(\log^{r+\ell} n)}$, in time $2^{\mathcal{O}(\log^{r+\ell} n)}n^{\mathcal{O}(1)}$. Then, for a good pair $(\mathcal{F}_1, \mathcal{F}_2)$ of graph families, we utilize the structure of the input graph G, to give a different approach to designing enumeration algorithms for $(\mathcal{F}_1, \mathcal{F}_2)$-separating families. Finally, we revisit the pair of graph families studied in Section 4.3.4. We find a $(\overline{\mathcal{D}}_0, \mathcal{D}_\ell))$-separating family of size $2^{\mathcal{O}(\ell \log^2 n)}$ in time $2^{\mathcal{O}(\ell \log^2 n)}n^{\mathcal{O}(1)}$.

4.4.1 Separating families for Sparse and Dense graphs

In this subsection, we give, for an n-vertex graph, an algorithm of running time $2^{\mathcal{O}(\log^3 n)}$, which enumerates a $(\mathcal{C}_1, \mathcal{I}_2)$-separating family of cardinality $2^{\mathcal{O}(\log^3 n)}$. In fact, the algorithm follows the steps of the protocol $\Pi_{1,2}$. It is known that we can enumerate a $(\mathcal{C}_1, \mathcal{I}_1)$-separating family $\mathcal{F}_{1,1}$ of cardinality $2^{(\log^2 n)/2}$ in time $2^{(\log^2 n)/2}n^{\mathcal{O}(1)}$ (see [Lovász 1990] and [Cygan 2013]). An enumeration algorithm for a $(\mathcal{C}_r, \mathcal{I}_\ell)$-separating family can be obtained by a direct generalisation of our algorithm for enumerating a $(\mathcal{C}_1, \mathcal{I}_2)$-separating family.

Lemma 4.3. *Every graph with n vertices has a $(\mathcal{C}_1, \mathcal{I}_2)$-separating family $\mathcal{F}_{1,2}$ of $2^{\mathcal{O}(\log^3 n)}$ sets. Moreover, such a family can be found in time $2^{\mathcal{O}(\log^3 n)}$.*

Proof. For an n-vertex graph H, we use $\mathcal{F}_{1,1}(H)$ to denote a $(\mathcal{C}_1, \mathcal{I}_1)$-separating family for H, of cardinality $2^{(\log^2 n)/2}$, enumerated in time $2^{(\log^2 n)/2}n^{\mathcal{O}(1)}$. We give a recursive algorithm to enumerate a $(\mathcal{C}_1, \mathcal{I}_2)$-separating family. Let G be the input graph on n

vertices. The steps are described in Algorithm 1. First, the set of all vertices with non-degree strictly less than $n/2$ is included as a set in our $(\mathcal{C}_1, \mathcal{I}_2)$-separating family $\mathcal{F}$. Then, if there is a vertex v in G_1 with non-degree at least $n/2$, then a $(\mathcal{C}_1, \mathcal{I}_2)$-separating family $\mathcal{S}_v$ is recursively found for $G - \overline{N}_G(v)$. This separating family is included in $\mathcal{F}$. On the other hand, if there is a vertex in G_2 with degree at least $n/2$, then a $(\mathcal{C}_1, \mathcal{I}_1)$-separating family, $\mathcal{F}_{1,1}(G[N_G(v)])$, is found for $G[N_G(v)]$. Also, a $(\mathcal{C}_1, \mathcal{I}_2)$-separating family $\mathcal{S}_v$ is recursively found for $G - N_G(v)$. In $\mathcal{F}$, we include all possible of unions, of a set taken from $\mathcal{S}_v$ and a set taken from $\mathcal{F}_{1,1}(G[N_G(v)])$. In the end, we return the family $\mathcal{F}$ as the potential separating family.

Algorithm 1: SepEnumeration(G)

Input: G
Output: A family $\mathcal{F}$ of vertex subsets
$S := \{v \mid \overline{d}_G(v) < n/2\}$
$\mathcal{F} := \{S\}$
for *each vertex* $v \in V(G)$ *with* $\overline{d}_G(v) \geq n/2$ **do**
  $\mathcal{S}_v :=$ SepEnumeration$(G - \overline{N}_G(v))$
  $\mathcal{F} := \mathcal{F} \cup \{A \mid A \in \mathcal{S}_v\}$
end
for *each vertex* $v \in V(G)$ *with* $d_G(v) \geq n/2$ **do**
  Compute $\mathcal{F}_{1,1}(G[N_G(v)])$
  $\mathcal{S}_v :=$ SepEnumeration$(G - N_G(v))$
  $\mathcal{F} := \mathcal{F} \cup \{A \cup B \mid A \in \mathcal{S}_v, B \in \mathcal{F}_{1,1}(G[N_G(v)])\}$
end
return $\mathcal{F}$.

We need to show that the family $\mathcal{F}$ is the required separating family. Suppose we are given a pair (G_1, G_2) with $V(G_1) \cap V(G_2) = \emptyset$. We prove that there is a separating set $A \in \mathcal{F}$ such that $V(G_1) \subseteq A$ and $V(G_2) \cap A = \emptyset$. We show this by induction on $|V(G)|$. In the base case, suppose that the graph G is an empty graph. It is trivially true that $V(G_1) \cap V(G_2) = \emptyset$ and $\emptyset$ is a separating set. In this case, $\mathcal{F} = \{\emptyset\}$ (because of Step 1). Now, assume that the algorithm is correct for any graph G with $|V(G)| < n$. Consider the case where $|V(G)| = n$. The following cases can occur:

1. Suppose there is a vertex $v \in V(G_1)$ with $\overline{d}_G(v) \geq n/2$. We know that $V(G_1) \cap \overline{N}_G(v) = \emptyset$. Thus, in the graph $G' = G[\overline{N}_G(v)]$, $\{\emptyset\}$ is a separating set for $G_1 \cap G'$ and $G_2 \cap G'$. Then, for any separating set A for $G'_1 = G_1 - \overline{N}_G(v) = G_1$ and $G'_2 = G_2 - \overline{N}_G(v)$ in the graph $G'' = G - \overline{N}_G(v)$, A is a separating set for G_1 and G_2. The graph G'' has strictly less number of vertices than G. By induction hypothesis, the family $\mathcal{S}_v$, computed in Step 1, contains a separating set A for G'_1 and G'_2. Then, A, included in $\mathcal{F}$ in Step 1, is the desired separating set for G_1 and G_2.

2. Suppose there is a vertex $v \in V(G_2)$ with $d_G(v) \geq n/2$. If B is a separating set for $G[N_G(v)] \cap G_1$ and $G[N_G(v)] \cap G_2$ in $G[N_G(v)]$, and A is a separating set for $G_1 - N_G(v)$ and $G_2 - N_G(v)$ in $G - N_G(v)$, then $A \cup B$ is a separating set for G_1 and G_2 in G. By Observation 4.2, $\omega(G[N_G(v)] \cap G_2) \leq 1$. We also know that $\alpha(G[N_G(v)] \cap G_1) \leq 1$. Hence, $\mathcal{F}_{1,1}(G[N_G(v)])$ contains a separating set B for $G[N_G(v)] \cap G_1$ and $G[N_G(v)] \cap G_2$. Let $G' = G - N_G(v)$. The graph G' has strictly less number of vertices than G. By induction hypothesis, the family $\mathcal{S}_v$ contains a

separating set A for $G_1 - N_G(v)$ and $G_2 - N_G(v)$. Then, $A \cup B$, included in Step 1, is the desired separating set for G_1 and G_2.

3. Suppose neither of the above conditions hold. Then every vertex $v \in V(G_1)$ has $\overline{d}_G(v) < n/2$, while every vertex $w \in V(G_2)$ has $d_G(w) < n/2$. Then by Observation 4.1, the set $S := \{v \mid \overline{d}_G(v) < n/2\}$ is a separating set for G_1 and G_2, which is included in $\mathcal{F}$ in Step 1.

Thus, we have proved the correctness of the algorithm. Notice that, whenever we make a recursive call, we reduce the vertex set of G to at most half. Let $S(n)$ be the size of the family $\mathcal{F}$ that is output. Then the recurrence formula $S(n) \leq n2^{(\log^2 n)/2} \cdot S(n/2) + 1$ holds. The recurrence for $S(n)$ solves to $2^{\mathcal{O}(\log^3 n)}$. A similar recurrence for the running time gives us the required running time bound. □

Corollary 4.6. *Every graph G with n vertices has a $(\mathcal{C}_2, \mathcal{I}_1)$-separating family of cardinality $2^{\mathcal{O}(\log^3 n)}$. Such a family can be enumerated in $2^{\mathcal{O}(\log^3 n)}$ time.*

Proof. For each $F \in \mathcal{F}_{1,2}$ we take $V(G) - F$ into $\mathcal{F}_{2,1}$. Consider induced subgraphs G_1, G_2 of G, such that $G_1 \in \mathcal{C}_2$ and $G_2 \in \mathcal{I}_1$. Then in $\overline{G}$, $\overline{G_2} \in \mathcal{C}_1$ and $\overline{G_1} \in \mathcal{I}_2$. If F is a separating set for $\overline{G_2}$ and $\overline{G_1}$, then $V(\overline{G_2}) \subseteq F$ and $V(\overline{G_1}) \cap F = \emptyset$. This implies that $V(G_1) \subseteq V(G) - F$ while $V(G_2) \cap (V(G) - F) = \emptyset$. Thus, $V(G) - F$ is a separating set for G_1 and G_2. Thus, we have constructed a $(\mathcal{C}_2, \mathcal{I}_1)$-separating family. □

We can enumerate a $(\mathcal{C}_r, \mathcal{I}_\ell)$-separating family of size $2^{\mathcal{O}(\log^{r+\ell} n)}$ using $(\mathcal{C}_{r-1}, \mathcal{I}_\ell)$- and $(\mathcal{C}_r, \mathcal{I}_{\ell-1})$-separating families, by generalising the above enumeration algorithm for the $(\mathcal{C}_1, \mathcal{I}_2)$-separating family. Thus, we get the following theorem.

Lemma 4.4. *For any $r, \ell \in \mathbb{N}$, every graph with n vertices has a $(\mathcal{C}_r, \mathcal{I}_\ell)$-separating family of cardinality $2^{\mathcal{O}(\log^{r+\ell} n)}$. Moreover, such a family can be enumerated in time $2^{\mathcal{O}(\log^{r+\ell} n)}$.*

Proof. We prove the Theorem by induction on $r + \ell$. Both $r, \ell > 0$. When $r + \ell = 2$, then $r = \ell = 1$ and, from [Cygan 2013], we get our desired separating family $\mathcal{F}_{1,1}$. When $r + \ell = 3$, either $r = 1, \ell = 2$ or $r = 2, \ell = 1$. From Lemma 4.3 and Corollary 4.6, we get the desired separating families.

We assume that we have the desired separating families when $r + \ell < m$, for some positive integer m. Now suppose we are given the pair r, ℓ such that $r + \ell = m$.

For an n-vertex graph H, and positive constants r', ℓ' such that $r' + \ell' < m$, we use $\mathcal{F}_{r',\ell'}(H)$ to denote a $(\mathcal{C}_{r'}, \mathcal{I}_{\ell'})$-separating family for H, of cardinality $2^{\mathcal{O}(\log^{r'+\ell'} n)}$, enumerated in time $2^{\mathcal{O}(\log^{r'+\ell'} n)} n^{\mathcal{O}(1)}$. By induction hypothesis, such a separating family can be found.

We give a recursive algorithm to enumerate a $(\mathcal{C}_r, \mathcal{I}_\ell)$-separating family. Let G be the input graph on n vertices. Our steps are described in Algorithm 2. First, the set of all vertices with non-degree strictly less than $n/2$ is included as a set in our $(\mathcal{C}_r, \mathcal{I}_\ell)$-separating family $\mathcal{F}$. Then, if there is a vertex v in G_1 with non-degree at least $n/2$, then a $(\mathcal{C}_{r-1}, \mathcal{I}_\ell)$-separating family, $\mathcal{F}_{r-1,\ell}(G[\overline{N}_G(v)])$, is found for $G[\overline{N}_G(v)]$. Also, a $(\mathcal{C}_r, \mathcal{I}_\ell)$-separating family $\mathcal{S}_v$ is recursively found for $G - \overline{N}_G(v)$. In $\mathcal{F}$, we include all possible of unions, of a set taken from $\mathcal{S}_v$ and a set taken from $\mathcal{F}_{r-1,\ell}(G[\overline{N}_G(v)])$. On the other hand, if there is a vertex in G_2 with degree at least $n/2$, then a $(\mathcal{C}_r, \mathcal{I}_{\ell-1})$-separating

Algorithm 2: SepEnumeration(G)

Input: G
Output: A family $\mathcal{F}$ of vertex subsets
$S := \{v \mid \overline{d}_G(v) < n/2\}$
$\mathcal{F} := \{S\}$
for *each vertex* $v \in V(G)$ *with* $\overline{d}_G(v) \geq n/2$ **do**
  Compute $\mathcal{F}_{r-1,\ell}(G[\overline{N}_G(v)])$
  $\mathcal{S}_v :=$ SepEnumeration$(G - \overline{N}_G(v))$
  $\mathcal{F} := \mathcal{F} \cup \{A \cup B \mid A \in \mathcal{S}_v, B \in \mathcal{F}_{r-1,\ell}(G[\overline{N}_G(v)])\}$
end
for *each vertex* $v \in V(G)$ *with* $d_G(v) \geq n/2$ **do**
  Compute $\mathcal{F}_{r,\ell-1}(G[N_G(v)])$
  $\mathcal{S}_v :=$ SepEnumeration$(G - N_G(v))$
  $\mathcal{F} := \mathcal{F} \cup \{A \cup B \mid A \in \mathcal{S}_v, B \in \mathcal{F}_{r,\ell-1}(G[N_G(v)])\}$
end
return $\mathcal{F}$.

family, $\mathcal{F}_{r,\ell-1}(G[N_G(v)])$, is found for $G[N_G(v)]$. Also, a $(\mathcal{C}_r, \mathcal{I}_\ell)$-separating family $\mathcal{S}_v$ is recursively found for $G - N_G(v)$. In $\mathcal{F}$, we include all possible of unions, of a set taken from $\mathcal{S}_v$ and a set taken from $\mathcal{F}_{r,\ell-1}(G[N_G(v)])$. In the end, we return the family $\mathcal{F}$ as the potential separating family.

We need to show that the family $\mathcal{F}$ is the required separating family. Suppose we are given a pair (G_1, G_2) with $V(G_1) \cap V(G_2) = \emptyset$. We prove that there is a separating set $A \in \mathcal{F}$ such that $V(G_1) \subseteq A$ and $V(G_2) \cap A = \emptyset$. We show this by induction on $|V(G)|$. In the base case, suppose that the graph G is an empty graph. It is trivially true that $V(G_1) \cap V(G_2) = \emptyset$ and $\emptyset$ is a separating set, and $\mathcal{F} = \{\emptyset\}$ in this case (because of Step 2). Now, assume that the algorithm is correct for any graph G with $|V(G)| < n$. Consider the case where $|V(G)| = n$. The following cases can occur:

1. Suppose there is a vertex $v \in V(G_1)$ with $\overline{d}_G(v) \geq n/2$. If B is a separating set for $G[\overline{N}_G(v)] \cap G_1$ and $G[\overline{N}_G(v)] \cap G_2$ in $G[\overline{N}_G(v)]$, and A is a separating set for $G_1 - \overline{N}_G(v)$ and $G_2 - \overline{N}_G(v)$ in $G - \overline{N}_G(v)$, then $A \cup B$ is a separating set for G_1 and G_2 in G. By Observation 4.2, $\alpha(G[\overline{N}_G(v)] \cap G_1) \leq r - 1$. We also know that $\omega(G[\overline{N}_G(v)] \cap G_2) \leq \ell$. Hence, $\mathcal{F}_{r-1,\ell}(G[\overline{N}_G(v)])$ contains a separating set B for $G[\overline{N}_G(v)] \cap G_1$ and $G[\overline{N}_G(v)] \cap G_2$. Let $G' = G - \overline{N}_G(v)$. The graph G' has strictly less number of vertices than G. By induction hypothesis, the family $\mathcal{S}_v$ contains a separating set A for $G_1 - \overline{N}_G(v)$ and $G_2 - \overline{N}_G(v)$. Then, $A \cup B$ is included in Step 2. This is the desired separating set for G_1 and G_2.

2. Suppose there be a vertex $v \in V(G_2)$ with $d_G(v) \geq n/2$. If B is a separating set for $G[N_G(v)] \cap G_1$ and $G[N_G(v)] \cap G_2$ in $G[N_G(v)]$, and A is a separating set for $G_1 - N_G[v]$ and $G_2 - N_G[v]$ in $G - N_G[v]$, then $A \cup B$ is a separating set for G_1 and G_2 in G. By Observation 4.2, $\omega(G[N_G(v)] \cap G_2) \leq \ell - 1$. We also know that $\alpha(G[N_G(v)] \cap G_1) \leq r$. Hence, $\mathcal{F}_{r,\ell-1}(G[N_G(v)])$ contains a separating set B for $G[N_G(v)] \cap G_1$ and $G[N_G(v)] \cap G_2$. Let $G' = G - N_G[v]$. The graph G' has strictly less number of vertices than G. By induction hypothesis, the family $\mathcal{S}_v$ contains a separating set A for $G_1 - N_G[v]$ and $G_2 - N_G[v]$. Then, $A \cup B$ is included in Step 2. This is the desired separating set for G_1 and G_2.

3. Suppose neither of the above conditions hold. Then every vertex $v \in V(G_1)$ has $\overline{d}_G(v) < n/2$, while every vertex $w \in G_2$ has $d_G(w) < n/2$. Then, by Observation 4.1, the set $S := \{v \mid \overline{d}_G(v) < n/2\}$ is a separating set for G_1 and G_2. This set is included in $\mathcal{F}$ in Step 2.

Thus, we have proved the correctness of the algorithm.

Notice that, whenever we make a recursive call, we reduce the vertex set of G to at least half. Let $S(n)$ be the size of the family $\mathcal{F}$ that is output. Then the recurrence formula $S(n) \leq n2^{c(\log^{r+\ell-1} n)} \cdot S(n/2) + 1$ holds, where c is a constant. The recurrence for $S(n)$ solves to $2^{\mathcal{O}(\log^{r+\ell} n)}$. A similar recurrence for the running time gives us the required upper bound on the running time of the algorithm. □

Lemma 4.4 and Corollary 4.2 gives us the following Corollary.

Corollary 4.7. *Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be a good pair of graph families. Then, there are constants r and ℓ, such that every n-vertex graph has a $(\mathcal{F}_1, \mathcal{F}_2)$-separating family of cardinality $2^{\mathcal{O}(\log^{r+\ell} n)}$ and it can be enumerated in time $2^{\mathcal{O}(\log^{r+\ell} n)}$.*

In fact, we obtain Theorem 4.4 from Lemma 4.4 and Corollary 4.2.

4.4.2 Separating families and parameterization

We give the proof of Theorem 4.5. We show that the upper bound obtained due to Corollary 4.7 can be improved if we use ideas from Parameterized Complexity, as we did for Theorem 4.2. This is extremely useful for designing FPT algorithms. To show this, we first prove the following lemma.

Lemma 4.5. *Let $(\mathcal{F}_1, \mathcal{F}_2)$ be a good pair of graph families. Given, as input, $G, S \subseteq V(G)$ and partition $V_1 \uplus V_2$ of $V(G) \setminus S$ such that $G[V_1] \in \mathcal{F}_1$ and $G[V_2] \in \mathcal{F}_2$, there is an algorithm to enumerate $(\mathcal{F}_1, \mathcal{F}_2)$-separating family $\mathcal{S}$ for G of cardinality $2^{\mathcal{O}(\log^c(|S|))}n^{\mathcal{O}(1)}$ in time $2^{\mathcal{O}(\log^c(|S|))}n^{\mathcal{O}(1)}$, where c is a constant.*

Proof. The proof of this Lemma follows the idea of the protocol built in Theorem 4.2. We know that $G[V_1] \in \mathcal{F}_1$ and $G[V_2] \in \mathcal{F}_2$. Since $(\mathcal{F}_1, \mathcal{F}_2)$ is a good pair of graph families, by Corollary 4.2, we know that there are constants r, ℓ, such that for any $G_1 \in \mathcal{F}_1$ and $G_2 \in \mathcal{F}_2$, $\omega(G_1) \leq r$ and $\alpha(G_2) \leq \ell$. Let us define $c = r + \ell$. By Lemma 4.4, the graph $G[S]$ has a $(\mathcal{F}_1, \mathcal{F}_2)$-separating family $\mathcal{S}'$ of cardinality $2^{\mathcal{O}(\log^{r+\ell} |S|)}$. Moreover, such a family can be enumerated in time $2^{\mathcal{O}(\log^{r+\ell}(|S|))}$. Now consider the following family.

$$\mathcal{S} = \left\{A \cup (V_1 \setminus S_1) \cup S_2 \mid A \in \mathcal{S}', \forall i \in \{1,2\} : S_i \subseteq V_i, |S_i| < R(r+1, \ell+1)\right\}$$

The cardinality of $\mathcal{S}$ is bounded by $2^{\mathcal{O}(\log^c(|S|))}n^{\mathcal{O}(2R(r+1,\ell+1))}$ and it can be enumerated in time $2^{\mathcal{O}(\log^c(|S|))}n^{\mathcal{O}(2R(r+1,\ell+1))}$. We show that $\mathcal{S}$ is indeed a $(\mathcal{F}_1, \mathcal{F}_2)$-separating family for G. Consider any disjoint vertex subsets U_1 and U_2 of $V(G)$ such that $G[U_1] \in \mathcal{F}_1$ and $G[U_2] \in \mathcal{F}_2$. We need to show that there is a set $T \in \mathcal{S}$ such that $U_1 \subseteq T$ and $T \cap U_2 = \emptyset$. Since the two families $\mathcal{F}_1$ and $\mathcal{F}_2$ are hereditary, $G[U_1 \cap S] \in \mathcal{F}_1$ and $G[U_2 \cap S] \in \mathcal{F}_2$. Since $\mathcal{S}'$ is a $(\mathcal{F}_1, \mathcal{F}_2)$-separating family for $G[S]$ there is a set $A \in \mathcal{S}'$ such that $S \cap U_1 \subseteq A$

and $(S \cap U_2) \cap A = \emptyset$. Since $G[U_1], G[V_1] \in \mathcal{F}_1$ and $G[U_2], G[V_2] \in \mathcal{F}_2$, by Corollary 4.3, we know that $|U_1 \cap V_2| < R(r+1, \ell+1)$ and $|U_2 \cap V_1| < R(r+1, \ell+1)$. Now consider the set $T = A \cup (V_1 \setminus (U_2 \cap V_1)) \cup (U_1 \cap V_2)$. Since $|U_1 \cap V_2| < R(r+1, \ell+1)$ and $|U_2 \cap V_1| < R(r+1, \ell+1)$, by the definition of $\mathcal{S}$, $T \in \mathcal{S}$. Notice that $U_1 \subseteq T$ and $U_2 \cap T = \emptyset$. Hence, $\mathcal{S}$ is a $(\mathcal{F}_1, \mathcal{F}_2)$-separating family for G. □

Lemma 4.5 gives us Theorem 4.5. Suppose there was an approximation algorithm $\mathcal{A}$ for $(\mathcal{F}_1, \mathcal{F}_2)$-Partition, where the approximation factor is defined by a computable function f depending only on the size of an optimal solution, and let the running time of $\mathcal{A}$ be $T(n)$ on an n-vertex input graph. Then, a $(\mathcal{F}_1, \mathcal{F}_2)$-seperating family, for G, of cardinality $2^{\mathcal{O}(\log^c f(\mathsf{opt}^G_{\mathcal{F}_1,\mathcal{F}_2}))} n^{\mathcal{O}(1)}$ can be enumerated in time $2^{\mathcal{O}(\log^c f(\mathsf{opt}))} n^{\mathcal{O}(1)}$, where c is the same constant as in Theorem 4.5. We would like to remark that the constant c in Theorem 4.5 is at most $r + \ell$.

4.4.3 Separating families when parameterized by degeneracy

In this subsection we consider the pair of graph families $(\mathcal{C}_1, \mathcal{D}_\ell)$, where $\mathcal{C}_1$ is the family of complete graphs. The protocol described in Section 4.3.4 shows that $D(\mathsf{GDISJ}_{G,\mathcal{C}_1,\mathcal{D}_\ell}) = \mathcal{O}(\ell \log^2 n)$. In this section, we give an enumeration algorithm for a $2^{\mathcal{O}(\ell \log^2 n)}$-sized $(\mathcal{C}_1, \mathcal{D}_\ell)$-separating family in time $2^{\mathcal{O}(\ell \log^2 n)} n^{\mathcal{O}(1)}$. The enumeration algorithm is described below.

Lemma 4.6. *For any constant $\ell \in \mathbb{N}$, there is an algorithm to enumerate a $(\mathcal{C}_1, \mathcal{D}_\ell)$-separating family for an n-vertex graph, of cardinality $n^{\mathcal{O}(\ell \log n)}$, in time $n^{\mathcal{O}(\ell \log n)}$.*

Proof. We give a recursive algorithm. Let G be a graph on n vertices. We describe the steps in Algorithm 3.

The algorithm runs for a recursion depth of at most j, for an input graph G. It recursively builds a family $\mathcal{F}$ of vertex subsets. First, we include $\emptyset$ in $\mathcal{F}$. Next, if there is a vertex v with non-degree at least $n/2$, then the algorithm recursively finds a family $\mathcal{S}_v$ of vertex sets of $G - \overline{N}_G(v)$, within a recursion depth of $j-1$. All sets in $\mathcal{S}_v$ are included in $\mathcal{F}$. On the other hand, suppose there is a vertex v with degree at least $n/2$. The family $\mathcal{F}_v$, of all 4ℓ-sized subsets of $N_G[v]$, is computed. Based on this, the family $\mathcal{F}'_v$, containing all possible sets $N_G[v] - F$ where $F \in \mathcal{F}_v$, is computed. Then, the algorithm recursively finds a family $\mathcal{S}_v$ of vertex sets of $G - N_G[v]$, within a recursion depth of $j-1$. For any set from $\mathcal{S}_v$ and any set from $\mathcal{F}'_v$, we put the union of the pair of sets into $\mathcal{F}$. Finally, let V' be the set of vertices in G that have non-degree strictly greater than $n/2$. The algorithm recursively finds a family of vertex sets of $G - V'$, within a recursion depth of $j-1$. All sets of this family is included in $\mathcal{F}$. With this, the computation of the family $\mathcal{F}$ is complete.

We show that the family $\mathcal{F}$, given as output in step 3 of $\mathsf{SepEnum}(G, 2\lceil \log |V(G)| \rceil)$, is the required separating family. In the proof, we have also used $\mathsf{SepEnum}(G, j)$ to denote the output family of the same problem instance.

First, we show the following Claim.

Claim 4.1. *Let G be a graph, and j be a positive integer. Then,* $\mathsf{SepEnum}(G, j-1) \subseteq \mathsf{SepEnum}(G, j)$.

Algorithm 3: $\mathsf{SepEnum}(G, j)$

Input: G, j
Output: A family $\mathcal{F}$ of vertex subsets
$\mathcal{F} = \{\emptyset\}$
if $j = 0$ **then**
  return $\mathcal{F}$
end
for *each vertex* $v \in V(G)$ *with* $\overline{d}_G(v) \geq n/2$ **do**
  $\mathcal{S}_v = \mathsf{SepEnum}(G - \overline{N}_G(v), j-1)$
  $\mathcal{F} = \mathcal{F} \cup \{A \mid A \in \mathcal{S}_v\}$
end
for *each vertex* $v \in V(G)$, $d_G(v) \geq n/2$ **do**
  $\mathcal{F}_v = \{F \mid F \subseteq N_G[v], v \in F, |F| \leq 4\ell, G[F] \in \mathcal{D}_\ell\}$
  $\mathcal{F}'_v = \{N_G[v] \setminus F \mid F \in \mathcal{F}_v\}$
  $\mathcal{S}_v = \mathsf{SepEnum}(G - N_G[v], j-1)$
  $\mathcal{F} = \mathcal{F} \cup \{A \cup B \mid A \in \mathcal{S}_v, B \in \mathcal{F}'_v\}$
end
$V' = \{v | \overline{d}_G(v) \geq n/2\}$
$\mathcal{F} = \mathcal{F} \cup \mathsf{SepEmuneration}(G - V', j-1)$
return $\mathcal{F}$

Proof. We prove this by induction on j. In the base case, $j = 1$. We know that when $j = 0$, $\mathsf{SepEnum}(G, 0) = \{\emptyset\}$. By definition, $\emptyset \in \mathsf{SepEnum}(G, 1)$. Therefore, the base case is true. By induction hypothesis, for all $1 \leq i < j$, let it be true that for any graph G, $\mathsf{SepEnum}(G, i-1) \subseteq \mathsf{SepEnum}(G, i)$.

Now, we deal with the case for j. Take a graph G, and a set $A \in \mathsf{SepEnum}(G, j-1)$. We show by case analysis, that A must also belong to $\mathsf{SepEnum}(G, j)$.

1. Let $A = \emptyset$. Then, by step 3, $A \in \mathsf{SepEnum}(G, j)$.

2. Let A be added to $\mathsf{SepEnum}(G, j-1)$ in step 3. By definition, there is a vertex $v \in V(G)$ with $\overline{d}_G(v) \geq n/2$ and $G' = G - \overline{N}_G(v)$ such that $A \in \mathsf{SepEnum}(G', j-2)$. By induction hypothesis, $A \in \mathsf{SepEnum}(G', j-1)$. Then, in step 3 of $\mathsf{SepEnum}(G, j)$, A is included.

3. Let A be added to $\mathsf{SepEnum}(G, j-1)$ in step 3. By definition, there is a vertex $v \in V(G)$, $d_G(v) \geq n/2$, a set $B' \in \mathcal{F}'_v$ (step 3), and a set $A' \in \mathsf{SepEnum}(G - N_G[v], j-2)$ such that $A = A' \cup B'$. By induction hypothesis, $A' \in \mathsf{SepEnum}(G - N_G[v], j-1)$. Then, in step 3 of $\mathsf{SepEnum}(G, j)$, A is included.

4. Let A be added to $\mathsf{SepEnum}(G, j-1)$ in step 3. Then, A belongs to $\mathsf{SepEnum}(G - V', j-2)$. By induction hypothesis, $A \in \mathsf{SepEnum}(G - V', j-1)$. Then, in step 3 of $\mathsf{SepEnum}(G, j)$, A is included.

Thus, we have proved the claim. □

Next, we show that for any graph G on n vertices, and a pair of induced subgraphs G_1, G_2 with $V(G_1) \cap V(G_2) = \emptyset$, $\mathsf{SepEnum}(G, \lceil \log |V(G)| \rceil + \lceil \log |V(G_2)| \rceil)$ contains a separating

set for G_1, G_2. We show this by double induction on $|V(G)|$ and $|V(G_2)|$. In the base case, suppose that either the graph G is an empty graph or the graph G_2 is an empty graph. Then, it is trivially true that $V(G_1) \cap V(G_2) = \emptyset$ and $\emptyset$ is a separating set. As the separating family $\mathcal{F}$ contains $\emptyset$, because of step 3, the induction hypothesis is trivially true. Now, assume that the algorithm is correct for $|V(G)| < n$, $|V(G_2)| < m \leq n$. Consider the cases where $|V(G)| = n$ or $|V(G_2)| = m$. The following cases can occur:

1. If there is a vertex $v \in V(C)$ such that $\overline{d}_G(v) \geq n/2$, we know that $\overline{N}_G(v) \cap V(G_1) = \emptyset$. Therefore, there is no vertex intersection of $G[\overline{N}_G(v)] \cap G_1$ and $G[\overline{N}_G(v)] \cap G_2$. It remains to separate $G'_1 = G_1$ and $G'_2 = G_2 - \overline{N}_G(v)$ in the graph $G' = G - \overline{N}_G(v)$. G' has a strictly smaller vertex set than G. Hence, by the induction hypothesis, it must be true that $\mathsf{SepEnum}(G', \lceil \log |V(G')| \rceil + \lceil \log |V(G'_2)| \rceil)$ contains a separating set A for G'_1, G'_2. Then, A is a separating set for G_1, G_2. This set is included in $\mathcal{F}$ due to step 3. The number of recursive steps to obtain this separating set is at most $1 + \lceil \log |V(G')| \rceil + \lceil \log |V(G'_2)| \rceil$. Since, the number of vertices is $V(G')$ is at most half of the number of vertices in $V(G)$, the number of recursive steps to obtain this separating set is at most $1 + (\lceil \log |V(G)| \rceil - 1) + \lceil \log |V(G_2)| \rceil \leq \lceil \log |V(G)| \rceil + \lceil \log |V(G_2)| \rceil$. Thus, we have proved the induction hypothesis in this case.

2. Suppose, there is a vertex $v \in V(G_2)$ such that $d_{G_2}(v) \leq 4\ell$ and $d_G(v) \geq n/2$. By definition of ℓ-degenerate graphs $G[N_{G_2}[v]]$ is also an ℓ-degenerate graph. By step 3, $N_{G_2}[v] \in \mathcal{F}_v$ and $N_G[v] \setminus N_{G_2}[v]$ belongs to $\mathcal{F}'_v$. We know that $N_G[v] \setminus N_{G_2}[v]$ is a separating set for $G[N_G[v]] \cap G_1$ and $G[N_G[v]] \cap G_2$. What remains is to separate $G'_1 = G_1 - N_G[v]$ and $G'_2 = G_2 - N_G[v]$ in the graph $G' = G - N_G[v]$. G' has a strictly smaller vertex set than G. Hence, by the induction hypothesis, it must be true that $\mathsf{SepEnum}(G', \lceil \log |V(G')| \rceil + \lceil \log |V(G'_2)| \rceil)$ contains a separating set A for G'_1, G'_2. Then, $A \cup (N_G[v] \setminus N_{G_2}[v])$ is a separating set for G_1, G_2. This set is included in $\mathcal{F}$ in step 3. The number of recursive steps to obtain this separating set is at most $1 + \lceil \log |V(G')| \rceil + \lceil \log |V(G'_2)| \rceil$. Since, the number of vertices in $V(G')$ is at most half of the number of vertices in $V(G)$, the number of recursive steps to obtain this separating set is at most $1 + (\lceil \log |V(G)| \rceil - 1) + \lceil \log |V(G_2)| \rceil \leq \lceil \log |V(G)| \rceil + \lceil \log |V(G_2)| \rceil$. Thus, we have proved the induction hypothesis in this case.

3. Suppose neither of the above conditions hold. This means that each vertex $v \in V(G_1)$ has $\overline{d}_G(v) < n/2$. Thus, a vertex w of $V(G_1) \cup V(G_2)$, that satisfies $\overline{d}_G(w) \geq n/2$, must come from $V(G_2)$. Now, a vertex u of degree at most 4ℓ in $V(G_2)$, satisfies $d_G(u) < n/2$. By Observation 4.1 and the definition in step 3, such a vertex u belongs to the vertex set V', while no vertex $v \in V(G_1)$ belongs to V'. If we want to find a separating set of G_1, G_2 of G, it is enough to find the separating set of $G'_1 = G_1 - V' = G_1$ and $G'_2 = G_2 - V'$ in $G' = G - V'$. G' has a strictly smaller vertex set than G. Hence, by the induction hypothesis, $\mathsf{SepEnum}(G', \lceil \log |V(G')| \rceil + \lceil \log |V(G'_2)| \rceil)$ contains a separating set A for G'_1, G'_2. Then, A is a separating set for G_1, G_2. Such a separating set is included in $\mathcal{F}$ in step 3. The number of recursive steps to obtain this separating set is at most $1 + \lceil \log |V(G')| \rceil + \lceil \log |V(G'_2)| \rceil$. By Observation 4.4, the number of vertices in $V(G'_2)$ is at most $\frac{1}{2}|V(G_2)|$. Thus, the number of recursive steps to obtain this separating set is at most $1 + \lceil \log |V(G)| \rceil + (\lceil \log |V(G_2)| \rceil - 1) \leq \lceil \log |V(G)| \rceil + \lceil \log |V(G_2)| \rceil$.

   Thus, we have proved the induction hypothesis in this case.

Thus, the induction hypothesis is correct.

Claim 4.2. *For any graph G, $\mathsf{SepEnum}(G, 2\lceil \log |V(G)| \rceil)$ is a $(\mathcal{C}_1, \mathcal{D}_\ell)$-separating family.*

Proof. Given a graph G, for induced subgraphs $G_1 \in \mathcal{C}_1$ and $G_2 \in \mathcal{D}_\ell$, we showed that the family $\mathsf{SepEnum}(G, \lceil \log |V(G)| \rceil + \lceil \log |V(G_2)| \rceil)$ is a family of vertex subsets that contains a separating set of G_1 and G_2. As G_2 is an induced subgraph of G, $\lceil \log |V(G)| \rceil + \lceil \log |V(G_2)| \rceil \leq 2\lceil \log |V(G)| \rceil$. From Claim 4.1, it is true that $\mathsf{SepEnum}(G, \lceil \log |V(G)| \rceil + \lceil \log |V(G_2)| \rceil) \subseteq \mathsf{SepEnum}(G, 2\lceil \log |V(G)| \rceil)$. Thus, for any pair of induced subgraphs $G_1 \in \mathcal{C}_1$ and $G_2 \in \mathcal{D}_\ell$, there must be a separating set of G_1 and G_2 in the family $\mathsf{SepEnum}(G, 2\lceil \log |V(G)| \rceil)$. Thus, $\mathsf{SepEnum}(G, 2\lceil \log |V(G)| \rceil)$ is a $(\mathcal{C}_1, \mathcal{D}_\ell)$-separating family. □

This proves the correctness of the algorithm.

Notice that, whenever we make a recursive call, we reduce the vertex set of G to at least half. Also, the recursion tree for this subproblem is allowed of depth strictly less than that of the parent problem. Let $S(n, j)$ be the size of the family $\mathcal{F}$ that is output for $\mathsf{SepEmuneration}(G, j)$. Then the recurrence formula $S(n, j) \leq n^\ell S(n/2, j-1)$ holds. In the base cases, $S(1, j) = 1$ and $S(n, 0) = 1$. The recurrence for $S(n, j)$ solves to $2^{\mathcal{O}(\ell j \log n)}$. Since we solve for $\mathsf{SepEmuneration}(G, 2\lceil \log |V(G)| \rceil)$, the size of the output family is $2^{\mathcal{O}(\ell \log^2 n)}$. A similar recurrence for the running time gives us the required upper bound on the running time of the algorithm. □

4.5 Applications in Parameterized and Exact Algorithms

In this section we relate the results obtained in previous sections to exact and FPT algorithms. The main result of this section is to show that the $(\mathcal{F}_1, \mathcal{F}_2)$-p-Partition problem is FPT. In fact, we propose an algorithm strategy that might result in faster running times than that of the best known algorithms for certain pairs $(\mathcal{F}_1, \mathcal{F}_2)$.

4.5.1 Combinatorial bounds and Exact Algorithms

In this part, we provide combinatorial bounds on the number of maximal induced subgraphs that have a vertex bipartition (A, B), where $G[A] \in \mathcal{F}_1$ and $G[B] \in \mathcal{F}_2$. We also give a strategy to design an enumeration algorithm for all such maximal induced subgraphs. Similarly, we can find the maximum(minimum) size of such an induced subgraph.

Theorem 4.7. *Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be a good pair of graph families.*

1. *For a graph G on n vertices, let $c_i^n, i \in \{1, 2\}$, be the size of the set of all maximal induced subgraphs that belong to $\mathcal{F}_i$. Then the size of the set of maximal induced subgraphs that have a vertex bipartition (A, B), where $G[A] \in \mathcal{F}_1$ and $G[B] \in \mathcal{F}_2$, is at most $2^{o(n)} \max\{c_1, c_2\}^n$.*

2. *Let $\mathcal{A}_i, i \in \{1, 2\}$, be an algorithm that takes in a graph G, of n vertices and m edges, and enumerates the set of all maximal induced subgraphs that belong to $\mathcal{F}_i$. Let the running time of $\mathcal{A}_i$, on a graph with n vertices and m edges, be $t_i(n + m)$.*

Then, given an n-vertex graph G, in time $2^{o(n)}\max\{t_1(n+m), t_2(n+m)\}$, we can enumerate the set of maximal induced subgraphs that have a vertex bipartition (A, B), where $G[A] \in \mathcal{F}_1$ and $G[B] \in \mathcal{F}_2$.

3. *Let $\mathcal{A}_i, i \in \{1, 2\}$, be an algorithm that takes in a graph G, of n vertices and m edges, and finds the maximum(minimum)-sized induced subgraphs that belongs to $\mathcal{F}_i$. Let the running time of $\mathcal{A}_i$, on a graph with n vertices and m edges, be $t_i(n+m)$. Then, given an n-vertex graph G, in time $2^{o(n)}\max\{t_1(n+m), t_2(n+m)\}$, we can obtain a maximum(minimum)-sized induced subgraph that has a vertex bipartition (A, B), where $G[A] \in \mathcal{F}_1$ and $G[B] \in \mathcal{F}_2$.*

Proof. First, we prove the combinatorial bound of (1). From Corollary 4.7, there is a $(\mathcal{F}_1, \mathcal{F}_2)$-separating family $\mathcal{F}$ for G, which is of size $2^o(n)$ and which can be found in time $2^{o(n)}n^{\mathcal{O}(1)}$. Consider a maximal induced subgraph H that has a vertex bipartition (A, B), where $G[A] \in \mathcal{F}_1$ and $G[B] \in \mathcal{F}_2$. By definition of $G[A]$ and $G[B]$, there is a separating set for these two graphs in $\mathcal{F}$. Such a separating set, say F, contains all vertices of A, and no vertices of B. Since A and B are disjoint, and H is a maximal induced subgraph, it must be the case that $G[A]$ is a maximal induced subgraph, belonging to $\mathcal{F}_1$, in $G[F]$. Similarly, $G[B]$ must be a maximal induced subgraph, belonging to $\mathcal{F}_2$, in $G[\overline{F}]$. Thus, to obtain an upper bound, it is enough to count the number of induced subgraphs of G, which have a vertex partition (A, B) such that $G[A] \in \mathcal{F}_1, G[B] \in \mathcal{F}_2$. Moreover, there is a set $F \in \mathcal{F}$ which contains all vertices of A and none of the vertices of B. For each $F \in \mathcal{F}$, there are $c_1^{|F|}$ maximal induced subgraphs of $G[F]$, that belong to $\mathcal{F}_1$. Similarly, there are $c_2^{|V(G)-F|}$ maximal induced subgraphs of $G - F$, that belong to $\mathcal{F}_2$. Thus, for this particular set F, there are $\max\{c_1, c_2\}^n$ induced subgraphs of G, that have a vertex bipartition (A, B), where $G[A] \in \mathcal{F}_1$, $G[B] \in \mathcal{F}_2$, $A \subseteq F$ and $B \subseteq V(G) \setminus F$. Going over all such sets, there are at most $2^{o(n)}\max\{c_1, c_2\}^n$ such induced subgraphs of G. Thus, there are at most $2^{o(n)}\max\{c_1, c_2\}^n$ required maximal induced subgraphs in G.

Next, we give a proof for (2). First, we enumerate a $(\mathcal{F}_1, \mathcal{F}_2)$-separating family $\mathcal{F}$ for G, using Corollary 4.7. For each set F in the separating family, we run $\mathcal{A}_1$ on $G[F]$ and $\mathcal{A}_2$ on $G[\overline{F}]$. Let $\mathcal{G}_i$ be the set of maximal induced subgraphs returned by the two algorithms. Then, we build a family $\mathcal{G}_F = \{G_1 \cup G_2 \mid G_1 \in \mathcal{G}_1, G_2 \in \mathcal{G}_2\}$. Let $\mathcal{G} = \bigcup_{F \in \mathcal{F}} \mathcal{G}_F$. The graphs in $\mathcal{G}$ are all graphs that have a vertex bipartition (A, B), where $G[A] \in \mathcal{F}_1$ and $G[B] \in \mathcal{F}_2$. We show that this family contains all such maximal graphs.

Consider any maximal induced subgraph H that has a vertex bipartition (A, B), where $G[A] \in \mathcal{F}_1$ and $G[B] \in \mathcal{F}_2$. By definition of $G[A]$ and $G[B]$, there is a separating set for these two graphs in $\mathcal{F}$. Such a separating set, say F, contains all vertices of A, and no vertices of B. Since A and B are disjoint, and H is a maximal induced subgraph, it must be the case that $G[A]$ is a maximal induced subgraph, belonging to $\mathcal{F}_1$, in $G[F]$. Similarly, $G[B]$ must be a maximal induced subgraph, belonging to $\mathcal{F}_2$, in $G[\overline{F}]$. By definition, the union of the two subgraphs must be contained in $\mathcal{G}_F$, and therefore, in $\mathcal{G}$.

Finally, we prune the collection $\mathcal{G}$, by removing any graph G whose vertex set in completely contained in the vertex set of a graph $H \in \mathcal{G}$. By definition of maximality, no maximal graph will be removed in this way. Since, all graphs in $\mathcal{G}$ had the desired vertex bipartition, it is only possible that after the pruning, only the maximal graphs remain. Thus, we are done.

Since the separating family is subexponential in size, the algorithm runs in time at most

$2^{o(n)} \max\{t_1(n+m), t_2(n+m)\}$.

Finally, we prove (3). First, we enumerate a $(\mathcal{F}_1, \mathcal{F}_2)$-separating family $\mathcal{F}$ for G, using Corollary 4.7. For each set F in the separating family, we run $\mathcal{A}_1$ on $G[F]$ and $\mathcal{A}_2$ on $G[\overline{F}]$. Let G_i^F be the maximum(minimum)-sized induced subgraphs returned by the two algorithms. Then, we build a family $\mathcal{G} = \{G_1^F \cup G_2^F \mid F \in \mathcal{F}\}$. The graphs in $\mathcal{G}$ are all graphs that have a vertex bipartition (A, B), where $G[A] \in \mathcal{F}_1$ and $G[B] \in \mathcal{F}_2$. We show that a maximum(minimum)-sized graph in $\mathcal{G}$, is a maximum(minimum)-sized induced subgraph that has a vertex bipartition (A, B), where $G[A] \in \mathcal{F}_1$ and $G[B] \in \mathcal{F}_2$.

On one hand, we show that there is a maximum(minimum)-sized induced subgraph, with the required vertex bipartization, in $\mathcal{G}$. Consider any maximum(minimum)-sized induced subgraph H that has a vertex bipartition (A, B), where $G[A] \in \mathcal{F}_1$ and $G[B] \in \mathcal{F}_2$. By definition of $G[A]$ and $G[B]$, there is a separating set for these two graphs in $\mathcal{F}$. Such a separating set, say F, contains all vertices of A, and no vertices of B. Since A and B are disjoint, and H is a maximum(minimum)-sized induced subgraph, it must be the case that $G[A]$ is a maximum(minimum)-sized induced subgraph, belonging to $\mathcal{F}_1$, in $G[F]$. Similarly, $G[B]$ must be a maximum(minimum)-sized induced subgraph, belonging to $\mathcal{F}_2$, in $G[\overline{F}]$. By definition, there is a graph $G_1^F \cup G_2^F \in \mathcal{G}$ of size $|G[A]| + |G[B]|$.

On the other hand, we show that a maximum(minimum)-sized induced subgraph in $\mathcal{G}$ has the required vertex bipartition. Let G' be a maximum(minimum)-sized graph in $\mathcal{G}$. By definition of the family, there is a family $F \in \mathcal{F}$ such that $G' = G_1^F \cup G_2^F$. This graph has a vertex bipartition $(V(G_1^F), V(G_2^F))$, where $G_1^F \in \mathcal{F}_1$ and $G_2^F \in \mathcal{F}_2$.

Thus, a maximum(minimum)-sized graph in $\mathcal{G}$, is a maximum(minimum)-sized induced subgraph that has a vertex bipartition (A, B), where $G[A] \in \mathcal{F}_1$ and $G[B] \in \mathcal{F}_2$.

Since the separating family is subexponential in size, the algorithm runs in time at most $2^{o(n)} \max\{t_1(n+m), t_2(n+m)\}$. □

Some concrete examples are given below.

Corollary 4.8. *Let $\mathcal{F}_2$ be the set of cliques. Let $\mathcal{F}_2$ be the set of all d-degenerate graphs, for a constant d. Then, given an n-vertex graph G, in time $(2-\epsilon_d)^{\mathcal{O}(n)}$, we can enumerate the set of maximal induced subgraphs that have a vertex bipartition (A, B), where $G[A] \in \mathcal{F}_1$ and $G[B] \in \mathcal{F}_2$. Here ϵ_d is a positive constant that depends only on d.*

Proof. Given an n-vertex graph, all maximal induced cliques can be enumerated in time $3^{\mathcal{O}(n/3)}$ [Moon]. Similarly, all maximal induced d-degenerate graphs can be enumerated in time $(2-\epsilon_d)^{\mathcal{O}(n)}$ [Pilipczuk 2012]. From Theorem 4.7, we get the desired result. □

Corollary 4.9. *Let $\mathcal{F}_1$ be the set of the complements of bipartite graphs, and $\mathcal{F}_2$ be the set of bipartite graphs. Then, given an n-vertex graph G, in time $1.7724^{\mathcal{O}(n)}$, we can enumerate the set of maximal induced subgraphs that have a vertex bipartition (A, B), where $G[A] \in \mathcal{F}_1$ and $G[B] \in \mathcal{F}_2$. We can also find the maximum-sized induced subgraph that has a vertex bipartition (A, B), where $G[A] \in \mathcal{F}_1$ and $G[B] \in \mathcal{F}_2$.*

Proof. Given an n-vertex graph, all maximal induced bipartite graphs can be enumerated in time $1.7724^{\mathcal{O}(n)}$ [Byskov 2004]. Our result follows from Theorem 4.7. □

Corollary 4.10. *Let $\mathcal{F}_1$ be the set of all cliques and let $\mathcal{F}_2$ be the set of all r-regular graphs, for a constant r. Then, when $r \geq 5$, given an n-vertex graph G, we can enumerate in time $2^{\mathcal{O}(1-\frac{1}{2r})}$, the set of maximal induced subgraphs that have a vertex bipartition (A, B), where $G[A] \in \mathcal{F}_1$ and $G[B] \in \mathcal{F}_2$. When $r \leq 4$, the running time of the enumeration algorithm is $1.7635^{\mathcal{O}(n)}$.*

Proof. It was shown in [Gupta 2012] that, for an n-vertex graph, when $r \leq 4$, all maximal induced r-regular graphs can be enumerated in time $1.7635^{\mathcal{O}(n)}$. On the other hand, when $r \geq 5$, all maximal induced r-regular graphs can be enumerated in time $2^{\mathcal{O}(1-\frac{1}{2r})}$. This, along with the result of [Moon] and Theorem 4.7, gives us the desired enumeration algorithm. □

Corollary 4.11. *Let $\mathcal{F}_1$ be the set of the cliques, and $\mathcal{F}_2$ be the set of forests. Then, given an n-vertex graph G, in time $1.8638^{\mathcal{O}(n)}$, we can enumerate the set of maximal induced subgraphs that have a vertex bipartition (A, B), where $G[A] \in \mathcal{F}_1$ and $G[B] \in \mathcal{F}_2$.*

Proof. By a result of [Fomin 2008], in an n-vertex graph, all maximal induced forests can be enumerated in time $1.8638^{\mathcal{O}(n)}$. This result, together with the result of [Moon] and 4.7, gives us the desired enumeration algorithm. □

4.5.2 Parameterized Algorithms

The question of what is the maximum size of an induced subgraph, that has a vertex bipartition (A, B) with $G[A] \in \mathcal{F}_1$ and $G[B] \in \mathcal{F}_2$, brings us to the question of how 'far' a graph is from becoming a graph with the desired bipartition. The $(\mathcal{F}_1, \mathcal{F}_2)$-p-Partition problem addresses this question. In this part, we look at this problem and a technique to solve this problem, when the pair of families are a good pair of families. The technique we use is an adaptation of the popular iterative compression technique.

First, we make an Observation.

Observation 4.5. *If an instance (G, k) is a* YES *instance of $(\mathcal{F}_1, \mathcal{F}_2)$-p-Partition, then for any induced subgraph $G' \leq_s G$, (G', k) is also a* YES *instance of $(\mathcal{F}_1, \mathcal{F}_2)$-p-Partition.*

Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be a good pair of graph families, and for any positive integer k, the families $\mathcal{F}_1 + kv$ and $\mathcal{F}_2 + kv$ have FPT recognition algorithms, that is, there are algorithms which take as input a graph G and an integer k, decides whether $G \in \mathcal{F}_i + kv, i \in \{1, 2\}$ and runs in time $f(k)|V(G)|^{\mathcal{O}(1)}$. For ease of notation, if $\mathcal{F}_1$ and $\mathcal{F}_2$ be a good pair of graph families, and the families $\mathcal{F}_1 + kv$ and $\mathcal{F}_2 + kv$ have FPT recognition algorithms, then we call $(\mathcal{F}_1, \mathcal{F}_2)$ an FPT-*good pair of families.*

We obtain a fast FPT algorithm for $(\mathcal{F}_1, \mathcal{F}_2)$-p-Partition by incorporating the iterative compression technique. For more details about the algorithmic technique of iterative compression we refer to the book (chapter 4 [Cygan 2015]).

Theorem 4.6. *Let $(\mathcal{F}_1, \mathcal{F}_2)$ be an* FPT*-good pair of families. Also, let $\mathcal{A}_1$ and $\mathcal{A}_2$ be the best recognition algorithms for $\mathcal{F}_1 + kv$ and $\mathcal{F}_2 + kv$ respectively. For an n-vertex input graph and non-negative integer k, let the running time of $\mathcal{A}_i, i \in \{1, 2\}$, be $T_i(n, k)$.*

Then $(\mathcal{F}_1, \mathcal{F}_2)$-p-Partition *on an instance* (G, k) *can be solved in time* $2^{\mathcal{O}(\log^c k)} n^{\mathcal{O}(1)} \cdot \max\{T_1(n,k), T_2(n,k)\}$.

Proof. Let (G, k) be an input instance. The algorithm is based on the iterative compression technique. Due to Observation 4.5, the iterative compression technique is meaningful for this problem. The iteration step is exactly as described in [Cygan 2015](chapter 4). This step has also been demonstrated in previous chapters of this thesis. The description of the compression problem and an algorithm to solve the same is given below.

The input of the compression problem is a graph G' and a vertex set $S \subseteq V(G')$, of size at most $k+1$. The set S satisfies the property that there is a partition $V_1 \uplus V_2$ of $V(G) \setminus S$ such that $G[V_1] \in \mathcal{F}_1$ and $G[V_2] \in \mathcal{F}_2$. The compression problem outputs YES if there is a vertex set S' of size at most k such that there is a partition $V_1' \uplus V_2'$ of $V(G) \setminus S'$ with $G[V_1'] \in \mathcal{F}_1$ and $G[V_2'] \in \mathcal{F}_2$. Otherwise, the output is NO. Lemma 4.5 can be used to solve the compression problem in time $2^{\mathcal{O}(\log^c k)} n^{\mathcal{O}(1)} \cdot 2k \max\{T_1(n,k), T_2(n,k)\}$.

By Lemma 4.5, we know that there is an enumeration algorithm which outputs a $(\mathcal{F}_1, \mathcal{F}_2)$-separating family $\mathcal{S}$ of cardinality $2^{\mathcal{O}(\log^c |S|)} n^{\mathcal{O}(1)}$ in time $2^{\mathcal{O}(\log^c |S|)} n^{\mathcal{O}(1)}$. Now for each $S \in \mathcal{S}$ and each pair of non-negative integers k_1, k_2 such that $k_1 + k_2 \leq k$, we run $\mathcal{A}_1$ on $(G[S], k_1)$ and $\mathcal{A}_2$ on $(G - S, k_2)$. We output YES if both $\mathcal{A}_1$ and $\mathcal{A}_2$ output YES. Otherwise our algorithm will output NO.

We show the correctness of the algorithm for the compression problem. Suppose the input instance is a YES instance. Let S' be a vertex set of size at most k such that there is a partition $V_1' \uplus V_2'$ of $V(G) \setminus S'$ with $G[V_1'] \in \mathcal{F}_1$ and $G[V_2'] \in \mathcal{F}_2$. Let $F \in \mathcal{S}$ be a separating set for the graphs $G[V_1']$ and $G[V_2']$. Also, let $|F \cap S| = k_1$ with $|(V(G) - F) \cap S| = k_1$. Since S is of size at most k, it must be the case that $k_1 + k_2 = |S| \leq k$. When we run $\mathcal{A}_\infty$ on $(G[F], k_1)$ and $\mathcal{A}_2$ on $(G - F, k_2)$, we output YES for both the subproblems. This means that we correctly detect that the input instance is a YES instance.

On the other hand, suppose there is a vertex set $F \in \mathcal{S}$ and a pair of integers k_1, k_2, with $k_1 + k_2 \leq k$, such that $\mathcal{A}_1$ on $(G[F], k_1)$ returns YES, and $\mathcal{A}_2$ on $(G - F, k_2)$ also returns YES. Then there is a k_1-sized vertex set S_1 in $G[F]$, such that $G[F] - S_1 \in \mathcal{F}_1$. Also, there is a k_2-sized vertex set S_2 in $G - F$ such that $(G - F) - S_2 \in \mathcal{F}_2$. This implies that there is a vertex set of size at most k whole deletion results in a graph G' with a vertex bipartition $V_1' \uplus V_2' = V(G')$ such that $G'[V_1'] \in \mathcal{F}_1$ and $G[V_2'] \in \mathcal{F}_2$. Thus, the input instance was a YES instance. Hence, the algorithm for the compression problem is correct.

The running time of the algorithm is bounded by $2^{\mathcal{O}(\log^c |S|)} n^{\mathcal{O}(1)} \cdot \max\{T_1(n,k), T_2(n,k)\} = 2^{\mathcal{O}(\log^c k)} n^{\mathcal{O}(1)} \cdot 2k \max\{T_1(n,k), T_2(n,k)\}$. Therefore, $(\mathcal{F}_1, \mathcal{F}_2)$-p-Partition, by iterative compression, can be solved in $2^{\mathcal{O}(\log^c k)} n^{\mathcal{O}(1)} \cdot \max\{T_1(n,k), T_2(n,k)\}$ time. □

We obtain several corollaries from Theorem 4.6. The Vertex $(2,2)$ Partization problem [Kolay 2015b, Baste 2015] is exactly $(\mathcal{F}_1, \mathcal{F}_2)$-p-Partition, when $\mathcal{F}_1$ is the set of bipartite graphs and $\mathcal{F}_2$ is the set of complements of bipartite graphs. The previous best running time for the problem is $3.3146^k |V(G)|^{\mathcal{O}(1)}$ [Kolay 2015b, Baste 2015] and has been described in Chapter 2.

Corollary 4.12. Vertex $(2,2)$ Partization *can be solved in time* $2.3146^k |V(G)|^{\mathcal{O}(1)}$.

Proof. Let $\mathcal{F}_1$ be the family of bipartite graphs and $\mathcal{F}_2$ be the family of complement

graphs of bipartite graphs. The pair $(\mathcal{F}_1, \mathcal{F}_2)$ is a good pair of families. By definition of the problem, recognition algorithms for $\mathcal{F}_1 + kv$ and $\mathcal{F}_2 + kv$ are equivalent to the Odd cycle transversal problem. The best known algorithm for Odd cycle transversal is $2.3146^k|V(G)|^{\mathcal{O}(1)}$ [Lokshtanov 2014]. Then, by Theorem 4.6, we have an algorithm for Vertex $(2,2)$ Partization, running in time $2.3146^k|V(G)|^{\mathcal{O}(1)}$. □

Corollary 4.13. *Consider* $(\mathcal{F}_1, \mathcal{F}_2)$-p-Partition *where* $\mathcal{F}_1$ *is the set of cliques, and* $\mathcal{F}_2$ *is the set of planar graphs. This problem can be solved in time* $2^{\mathcal{O}(k \log k)}|V(G)|^{\mathcal{O}(1)}$.

Proof. The pair $(\mathcal{F}_1, \mathcal{F}_2)$ is a good pair of families. A recognition algorithm for $\mathcal{F}_1 + kv$ is equivalent to solving Vertex Cover in the complement graph. A recognition algorithm for $\mathcal{F}_2 + kv$ is equivalent to the Planar Vertex Deletion problem. The best known algorithm for Vertex Cover is $1.2738^k|V(G)|^{\mathcal{O}(1)}$ [Chen 2010], while that for Planar Vertex Deletion is $2^{\mathcal{O}(k \log k)}|V(G)|^{\mathcal{O}(1)}$ [Jansen 2014]. Then, by Theorem 4.6, we have an algorithm for the given problem, running in time $2^{\mathcal{O}(k \log k)}|V(G)|^{\mathcal{O}(1)}$. □

Corollary 4.14. *Consider* $(\mathcal{F}_1, \mathcal{F}_2)$-p-Partition *where* $\mathcal{F}_1$ *is the set of cliques, and* $\mathcal{F}_2$ *is the set of triangle-free graphs. This problem can be solved in time* $2.270^k|V(G)|^{\mathcal{O}(1)}$.

Proof. The pair $(\mathcal{F}_1, \mathcal{F}_2)$ is a good pair of families. A recognition algorithm for $\mathcal{F}_1 + kv$ is equivalent to solving Vertex Cover in the complement graph. A recognition algorithm for $\mathcal{F}_2 + kv$ is equivalent to the Triangle-free Vertex Deletion problem. The best known algorithm for Vertex Cover is $1.2738^k|V(G)|^{\mathcal{O}(1)}$ [Chen 2010], while that for Triangle-free Vertex Deletion is $2.270^k|V(G)|^{\mathcal{O}(1)}$ [Niedermeier 2003]. Then, by Theorem 4.6, we have an algorithm for the given problem, running in time $2.270^k|V(G)|^{\mathcal{O}(1)}$. □

Corollary 4.15. *Consider* $(\mathcal{F}_1, \mathcal{F}_2)$-p-Partition *where* $\mathcal{F}_1$ *is the set of cliques, and* $\mathcal{F}_2$ *is the set of forests. This problem can be solved in time* $3.6181^k|V(G)|^{\mathcal{O}(1)}$.

Proof. The pair $(\mathcal{F}_1, \mathcal{F}_2)$ is a good pair of families. A recognition algorithm for $\mathcal{F}_1 + kv$ is equivalent to solving Vertex Cover in the complement graph. A recognition algorithm for $\mathcal{F}_2 + kv$ is equivalent to the Feedback Vertex Set problem. The best known algorithm for Vertex Cover is $1.2738^k|V(G)|^{\mathcal{O}(1)}$ [Chen 2010], while that for Feedback Vertex Set is $3.619^k|V(G)|^{\mathcal{O}(1)}$ [Kociumaka 2014]. Then, by Theorem 4.6, we have an algorithm for the given problem, running in time $3.619^k|V(G)|^{\mathcal{O}(1)}$. □

4.6 Chapter Summary

In this chapter, we studied the communication complexity of the function $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$, which is a generalisation of CIS_G. We also introduced ideas from parameterized complexity in the computation of upper bounds for the communication complexity of $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$. Finally, we obtained separating families for good pairs of families, and used them to give combinatorial bounds, exact algorithms and FPT algorithms. An important question here is to see if the lower bounds for CIS_G can be used to obtain non-trivial lower bounds for $\mathsf{GDISJ}_{G,\mathcal{F}_1,\mathcal{F}_2}$, when $(\mathcal{F}_1, \mathcal{F}_2)$ is a good pair of graph families. Also, it would be interesting to study the upper bounds for the communication complexity of these functions in terms of other relevant parameters of the input.

Chapter 5

Quick but Odd Growth of Cacti

5.1 Introduction

In the field of parameterized graph algorithms, vertex (edge) deletion (addition, editing) problems constitute a considerable fraction. In particular, let $\mathcal{F}$ be a family of graphs. Given an input graph G and a positive integer k, testing whether G has a k-sized subset of vertices (edges) S, such that $G - S$ belongs to $\mathcal{F}$, is a prototype vertex (edge) deletion problem. Many well known problems in parameterized complexity can be phrased in this language. For example, if $\mathcal{F}$ is a family of edgeless graphs, or forests or bipartite graphs, then the vertex deletion problems to convert the input graph into a graph in $\mathcal{F}$ are Vertex Cover, Feedback Vertex Set, and Odd Cycle Transversal, respectively. Most of these problems are NP-complete due to a classic result by Lewis and Yannakakis [Lewis 1980], and naturally a candidate for parameterized study (with respect to solution size). Vertex Cover, Feedback Vertex Set and Odd Cycle Transversal are some of the most well studied problems in the domain of parameterized complexity. These problems have led to identification of several new techniques and ideas in the field.

Recent years have seen a plethora of results around vertex and edge deletion problems, in the domain of parameterized complexity [Cao 2015a, Cao 2015b, Fomin 2012, Fomin 2013, Giannopoulou 2015, Joret 2014, Kim 2015]. All the problems dealt with in the previous chapters, of Part I, are vertex or edge deletion problems. In this chapter, we continue this line of research and study two vertex deletion problems. In particular, we study the problem of deleting vertices to get a cactus or an odd cactus graph. A graph H is called a *cactus* graph if H is connected and every pair of cycles in H intersect on at most one vertex. Furthermore, a cactus graph H is called an *odd cactus* graph, if every cycle of H is of odd length. A graph is called a forest of cacti if every component of the graph is a cactus graph. Let us denote by $\mathcal{C}$ and $\mathcal{C}_{\mathsf{odd}}$, the families of forests of cacti and forests of odd cacti, respectively. The vertex deletion problems corresponding to $\mathcal{C}$ and $\mathcal{C}_{\mathsf{odd}}$ are called Diamond Hitting Set and Even Cycle Transversal, respectively. It is important to note that the name of the problem, of deleting vertices to get into $\mathcal{C}_{\mathsf{odd}}$, is Even Cycle Transversal, because it is equivalent to deleting a k-sized subset S such that $G - S$ does not have any *cycle of even length*. More precisely, we study the following problems in the realm of parameterized complexity.

Even Cycle Transversal	**Parameter:** k
Input: An undirected graph G and a positive integer k.	
Question: Does there exist a vertex subset S of size at most k such that $G-S \in \mathcal{C}_{\mathsf{odd}}$?	

Diamond Hitting Set	**Parameter:** k
Input: An undirected graph G and a positive integer k.	
Question: Does there exist a vertex subset S of size at most k such that $G - S \in \mathcal{C}$?	

In this chapter, we work on multigraphs, which are graphs where two vertices may have parallel edges between them. For ease of notation, we refer to multigraphs as graphs. While Odd Cycle Transversal is one of the most well studied problem in the realm of parameterized complexity, there is only one article about Even Cycle Transversal in the literature. The structure of the graph without even cycles, or without cycles of length 0 modulo some positive integer p, is simple. Thomassen showed that such graphs have treewidth at most $f(p)$ [Thomassen 1988]. Misra et al. [Misra 2012] used the structural properties of an odd-cactus graph to design an algorithm for Even Cycle Transversal with running time $50^k n^{\mathcal{O}(1)}$. They also give an $\mathcal{O}(k^2)$ kernel for the problem. On the other hand, the family $\mathcal{C}$ of forests of cacti can be characterized by a single excluded minor. In particular, let Θ be a graph on two vertices that have three parallel edges, then a graph H belongs to $\mathcal{C}$ if and only if H does not contain Θ as a minor. Since Θ is a connected planar graph we obtain a $c^k n^{\mathcal{O}(1)}$ time algorithm as a corollary to the main results in [Fomin 2012, Joret 2014, Kim 2015]. However, we are not aware of exact value of c as all these algorithms use a protrusion subroutine [Bodlaender 2009]. The problem also has $\mathcal{O}(k^2)$ kernel [Fomin 2011]. In this chapter, we give the following algorithm for these problems.

Theorem 5.1. *There are randomized algorithms for* Diamond Hitting Set *and* Even Cycle Transversal *with worst case running time* $12^k n^{\mathcal{O}(1)}$*, where* n *and* m *are the number of vertices and edges in the input graph, respectively. The algorithm outputs* NO *if the input is a* NO *instance and for a* YES *instance, with probablity at least* $1 - \frac{1}{e}$ *returns a solution.*

Our Methods. Our algorithms use the same methodology that is used for the $4^k n^{\mathcal{O}(1)}$ time algorithm for Feedback Vertex Set [Becker 2000], and its generalisation to Planar $\mathcal{F}$ Deletion [Fomin 2012]. In both our algorithms, we start by applying some reduction rules to the given instance. After this, we show that the number of edges incident to any solution S of our problems, is a constant fraction to the total number of edges in the graph. This counting lemma is our main technical contribution. We also observe that the analysis for the counting lemma is tight for an infinite family of graphs, and thus the analysis of our randomized algorithms can not be improved. It is in the same spirit as finding an infinite family of instances for which an approximation algorithm achieves its approximation ratio.

To apply our reduction rules in a way that the ratio between the number of edges incident with a particular solution S of the problem and the total number of edges in the input graph is as small as possible, we study a more general problem than Even Cycle Transversal, which we call Parity Even Cycle Transversal. In this problem we are given a graph G and a weight function $w : E(G) \to \{0, 1\}$. The objective is to delete a subset S of vertices of size at most k such that in $G - S$ there is no cycle whose weight sum is even. Observe that if w assigns one to every edge then it is same as Even Cycle Transversal.

We conclude the introduction by noting that Diamond Hitting Set and Even Cycle Transversal admit approximation algorithms with factor 9 and 10 respectively [Fiorini 2010, Misra 2012]. The results of this chapter are from [Kolay 2015a].

5.2 Preliminaries

In this chapter, we use multigraphs, where two vertices can have more than one edge between them. The vertex set and edge set of a multigraph G, are denoted by $V(G)$ and $E(G)$ respectively. An edge, of the multigraph, between two vertices $u, v \in V(G)$ is denoted by (u, v), while a path between u, v is denoted by $[u, v]$. If a sequence of vertices $v_1, \ldots, v_t$ or edges $e_1, \ldots e_t$ form a path, then too we denote this path by $[v_1, \ldots, v_t]$ and $[e_1, \ldots e_t]$ respectively. Similarly, a cycle on vertices $\{v_1, v_2, \ldots, v_t\}$ is denoted as $[v_1, v_2, \ldots, v_t, v_1]$. For ease of notation, we will be referring to a multigraph as a graph. The notation used for edges and paths in a multigraph is restricted to this chapter only. A few other operations and functions on edges of a multigraph are given below. Given two subsets $V_1, V_2 \subseteq V(G)$, $E(V_1, V_2)$ denotes the set of edges in $E(G)$ that have one end point in V_1 and the other in V_2. For a vertex $v \in V(G)$ and subset $V' \subseteq V(G) \setminus \{v\}$ we use $E(v, V')$ to denote the edge set $E(\{v\}, V')$. The subdivision of an edge $e = (u, v)$ of a graph G results in a graph G', which contains a new vertex w, and where the edge e is replaced by two new edges (u, w) and (w, v). A graph $\hat{G}$ is a subdivision of a graph G if there is a sequence of graphs $\{G_1, G_2, \ldots, G_t\}$, with $G_1 = G$ and $G_t = \hat{G}$, where for each $1 < i \leq t$, G_i is obtained by the subdivision of an edge of G_{i-1}.

A *block* of a graph G is a maximal biconnected subgraph of G.

Definition 5.1 (Block-Cut Vertex Tree)**.** *Let G be a connected graph, C be the set of cut vertices of G and $\mathcal{B}$ be the set of blocks of G. The block-cut vertex tree H of G has vertex set $C \cup \mathcal{B}$ and $E(H) = \{(c, B) \mid c \in C, B \in \mathcal{B}, c \in V(B)\}$.*

In fact we can show that block-cut vertex tree of a graph is indeed a tree [Diestel 2012]. Now we explain how to construct a *block decomposition tree* of a connected graph. Let H be a block-cut vertex tree of a connected graph G. Let C be the set of cut vertices of G and $\mathcal{B}$ be the set of blocks of G. We arbitrarily root the tree H at a vertex B_r, where $B_r \in \mathcal{B}$. Now, a block decomposition tree $\mathcal{T}$ of G has vertex set $\mathcal{B}$, and $(B_1, B_2) \in E(\mathcal{T})$ if $V(B_1) \cap V(B_2) \neq \emptyset$ (in other words B_1 and B_2 share a cut vertex of G) and B_1 is an ancestor of B_2 in H. In other words, $\mathcal{T}$ is obtained from H by contracting the set of edges $\{(c, B) \mid c \in C, B \in \mathcal{B}, B \text{ is the parent of } c \text{ in } H\}$. Thus, $\mathcal{T}$ must be a tree. See Figure 5.1 for an illustration of block decomposition tree of a graph. A block decomposition tree of a graph can be built in polynomial time.

Lemma 5.1. *Let T be a tree. Let $V_1 = \{v \in V(T) \mid d_T(v) = 1\}$, $V_2 = \{v \in V(T) \mid d_T(v) = 2\}$ and $V_3 = \{v \in V(T) \mid d_T(v) \geq 3\}$. Then $\sum_{v \in V_3} d_T(v) \leq 3|V_1|$.*

Proof. We know that $|V(T)| = |V_1| + |V_2| + |V_3|$. Also, $\Sigma_{v \in V(T)} d_T(v) = 2|E(T)| = 2(|V(T)| - 1)$. Now, $\Sigma_{v \in V(T)} d_T(v) = \Sigma_{v \in V_1} d_T(v) + \Sigma_{v \in V_2} d_T(v) + \Sigma_{v \in V_3} d_T(v) \geq |V_1| + 2|V_2| + 3|V_3|$. Using the two equations we get that $|V_3| \leq |V_1| - 2 \leq |V_1|$. This also means, $\Sigma_{v \in V_3} d_T(v) = 2(|V_1| + |V_2| + |V_3| - 1) - (|V_1| + 2|V_2|) \leq |V_1| + 2|V_3|$. Using the bound of $|V_3|$, $\Sigma_{v \in V_3} d_T(v) \leq 3|V_1|$. □

Definition 5.2. *A cactus graph is a connected graph where any two cycles have at most one vertex in common. Equivalently, every edge of the graph belongs to at most one cycle. Another equivalent definition is that any block of a cactus graph can be either a cycle or an edge. A graph where every component is a cactus graph is called a forest of cacti.*

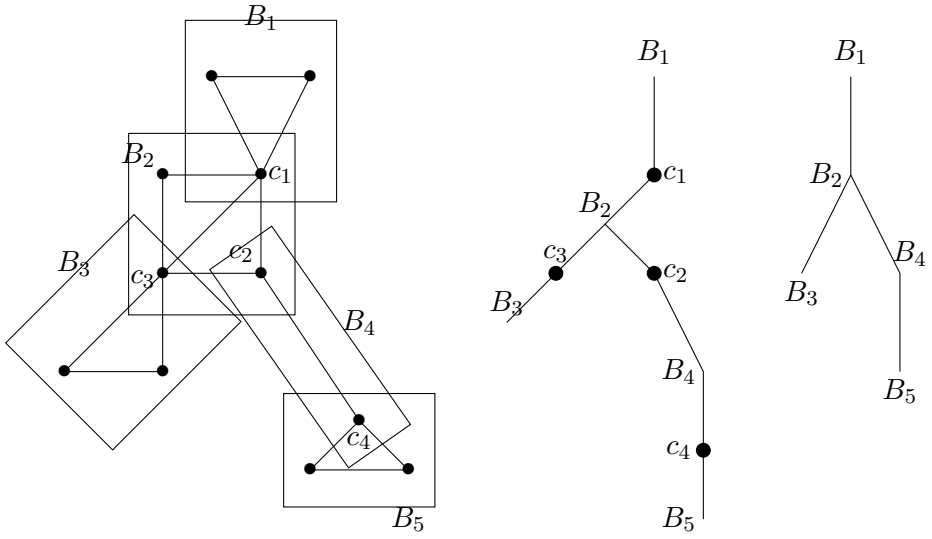


Figure 5.1: Left: a graph G with blocks B_1, B_2, B_3, B_4 and B_5. The cut vertices in G are c_1, c_2, c_3 and c_4; Middle: the block-cut vertex tree H of G; Right: a block decomposition tree $\mathcal{T}$ of G constructed from H rooted at B_1.

Definition 5.3. *Let Θ be a graph on a pair of vertices $\{u, v\}$ that have 3 parallel edges between them. A graph is called a diamond graph if it is obtained by a number of subdivisions of Θ.*

The following Proposition characterizes the class of forests of cacti.

Proposition 5.1. *A graph is a forest of cacti if and only if it does not have a diamond as a subgraph.*

The definition of diamond graphs and the characterization of forests of cacti have been taken from [Fiorini 2010]. We deviate a little from [Diestel 2012] on the notations for multigraphs.

5.3 Counting Lemma

In this section, we consider a graph G which has a set S, the deletion of which results in a cactus graph. Moreover, each vertex of the cactus graph $G - S$ has at least three distinct neighbours in G or shares at least two edges with S. Then, it is possible to bound the number of edges in $E(G - S)$ by the number of edges in $E(S, V(G) \setminus S)$. In fact, we exhibit a family of graphs where this bound is tight, up to a constant difference.

Lemma 5.2. *Let G be a graph and $S \subseteq V(G)$ such that $G - S$ is a cactus graph and for all $v \in V(G) \setminus S$ one of the following two conditions holds:*

1. *v has at least 3 distinct neighbours in G, or*
2. *there are at least two edges in $E(v, S)$*

Then $|E(G-S)| \leq 5|E(S, V(G) \setminus S)|$.

Proof. Let $G' = G - S$. We know that G' is a cactus graph. Let $\mathcal{T}$ be the block decomposition tree of G' rooted at a vertex of degree one. Throughout the proof, for a block X of G', we represent the corresponding vertex in $\mathcal{T}$ as t_X. Let $B = E(G')$ and $C = E(S, V(G) \setminus S)$. We need to show that $|B| \leq 5|C|$.

We first define some notations. Let X be a block of size at most 2 (an edge or a cycle of length 2) in G' such that t_X has only one child t_Y, which is a leaf node in $\mathcal{T}$. Then we say that the blocks X and Y together form a *super block*. If blocks X and Y form a super block Z, where t_Y is a leaf node, then by parent of the super block Z, we mean the parent of t_X in $\mathcal{T}$. All other blocks, which are not part of any super block, are called *normal blocks*. By *size* of a (super/normal) block Z, denoted by $\mathsf{size}(Z)$, we mean the number of edges in the block Z. To bound the number of edges in G', it is enough to bound the total number of edges in super blocks and normal blocks. Let $\mathcal{B}_\ell$ be the set containing all super blocks and normal blocks which correspond to leaves in $\mathcal{T}$. Let $\mathcal{B}_n$ be the set of normal blocks which are not part of $\mathcal{B}_\ell$. We define B_ℓ as the set of edges in the (normal/super) blocks which are part of $\mathcal{B}_\ell$, and B_n as the set of edges in the normal blocks which are part of $\mathcal{B}_n$. To bound the cardinality of B, it is enough to bound the cardinality of B_ℓ and B_n, individually. We partition the edges in C as follows. We say an edge $e \in C$ is incident to a (super/normal) block Z if it is incident to a vertex u in Z, which is not the cut vertex shared with the parent of Z. We use E_Z to denote the set of edges in C, which are incident to the (super/normal) block Z. Let C_ℓ be the set of edges in C which are incident to (super/normal) blocks in $\mathcal{B}_\ell$. Similarly, let C_n be the set of edges in C which are incident to blocks in $\mathcal{B}_n$. Let r_i be the number of blocks of size i in $\mathcal{B}_\ell$. Let $B_\ell^{(i)}$ be the set of edges in blocks of size i in $\mathcal{B}_\ell$. Let $C_\ell^{(i)}$ be the set of edges in C_ℓ which are incident to blocks of size i in $\mathcal{B}_\ell$. Notice that $B_\ell = \biguplus_i B_\ell^{(i)}$ and $C_\ell = \biguplus_i C_\ell^{(i)}$.

Claim 5.1. $r_i \leq \frac{|C_\ell^{(i)}|}{2}$ *for* $i \leq 4$ *and* $r_i \leq \frac{|C_\ell^{(i)}|}{i-3}$ *for* $i \geq 5$.

Proof. **Bound on r_1.** Let X be a block of size one in $\mathcal{B}_\ell$. That is, the block X is a single edge (x, y) and there is a vertex in $\{x, y\}$ which has degree one in G'. Let x be the degree one vertex. By our assumption at least 2 edges in $C_\ell^{(1)}$ are incident on x. This implies that $|E_X| \geq 2$. Thus, $|C_\ell^{(1)}| = \sum_{\{X:\mathsf{size}(X)=1\}} |E_X| \geq 2r_1$. Hence $r_1 \leq \frac{|C_\ell^{(1)}|}{2}$.

Bound on r_2. Let X be a block of size two in $\mathcal{B}_\ell$. If X is a normal block, then the block X is a cycle $[y, x, y]$ of length 2. Since X is leaf block, there is a vertex in X which is not a cut vertex in G'. Let x be the vertex in X such that x is not a cut vertex. This implies that $N_{G'}(x) = \{y\}$. Thus, by our assumption, either $|E(x, S)| \geq 2$ or x has two neighbours in S. In either case, $|E(x, S)| \geq 2$. That is, $|E_X| \geq 2$. If X is a super block, then X consists of two blocks Y and Z of size 1 each, such that t_Y has only one child t_Z and t_Z is a leaf node in $\mathcal{T}$. Let $Z = (x, y)$ be such that x has degree one in G'. Thus, by our assumption, we can conclude that $|E(x, S)| \geq 2$. That is, $|E_X| \geq 2$. Thus, we know that $|C_\ell^{(2)}| = \sum_{\{X:\mathsf{size}(X)=2\}} |E_X| \geq 2r_2$. Hence, $r_2 \leq \frac{|C_\ell^{(2)}|}{2}$.

Bound on r_3. Let X be a (super/normal) block of size three in $\mathcal{B}_\ell$. That is, either the block X is a cycle $[x, y, z, x]$ of length 3, or it is a super block consisting of two blocks, where one of them is a cycle of length 2 and the other is an edge. If X is a cycle $[x, y, z, x]$,

then t_X is a leaf in $\mathcal{T}$. Let z be the only cut vertex in $\{x, y, z\}$. This implies that the degrees of x and y are exactly 2 in G'. Thus, by our assumption, $|E(x, S)| \geq 1$ and $|E(y, S)| \geq 1$. This implies that $|E_X| \geq 2$.

Suppose X is a super block. Then X consists of a cycle $[x, y, x]$ and an edge (y, z). In this case, only one vertex, either x or z, will be shared with the parent of X and all other vertices will not have a neighbour in $G' - X$. Suppose x is the shared vertex with the parent of the block X. Then the number of distinct neighbours of y and z are exactly 2 and 1 respectively in G'. This implies that $|E(y, S)| \geq 1$ and $|E(z, S)| \geq 2$. Consequently, $|E_X| \geq 3$. By a similar argument, we can show that if z is the shared vertex of the super block X with its parent, then $|E_X| \geq 3$. Thus, $|C_\ell^{(3)}| = \sum_{\{X:\mathsf{size}(X)=3\}} |E_X| \geq 2r_3$. Hence, $r_3 \leq \frac{|C_\ell^{(3)}|}{2}$.

Bound on r_4. Let X be a (super/normal) block of size four in $\mathcal{B}_\ell$. That is, either the block X is a cycle of length 4 or it is a super block consisting of two blocks. If X is a cycle of length 4, then t_X is a leaf in $\mathcal{T}$. This implies that the degree of every vertex in X, except the cut vertex shared with the parent block, is exactly 2 in G'. This implies that $|E_X| \geq 3$.

Suppose X is a super block consisting of two blocks Y and Z, where the size of Y is at most 2 and t_Z is a leaf node in $\mathcal{T}$. If $\mathsf{size}(Y) = 1$, then Z is a cycle of length 3. This implies that at least two vertices in Z have degree exactly 2 in G'. Thus, by our assumption, $|E_Z| \geq 2$ and this implies that $|E_X| \geq 2$.

If $\mathsf{size}(Y) = 2$, then both Y and Z are cycles of length 2. Let x, y, x be the block Y and y, z, y be the block Z. Thus, the number of distinct neighbours of y and z in G' is 2 and 1 respectively. By our assumption, this implies that $|E(y, S)| \geq 1$ and $|E(z, S)| \geq 2$. Thus, $|E_X| \geq 3$. Hence, we conclude that $|C_\ell^{(4)}| = \sum_{\{X:\mathsf{size}(X)=4\}} |E_X| \geq 2r_4$. This means, $r_4 \leq \frac{|C_\ell^{(4)}|}{2}$.

Bound of r_i for $i \geq 5$. Let X be a (super/normal) block of size at least five in $\mathcal{B}_\ell$. That is, either the block X is a cycle of length i, or it is a super block consisting of two blocks Y and Z such that Z is a cycle of length at least $i - 2$ and t_Z is a leaf in $\mathcal{T}$. In either case, X contains at least $i - 3$ vertices (excluding the cut vertex shared with the parent block) having exactly 2 distinct neighbours in G'. This implies that $|E_X| \geq i - 3$. Hence, it must be true that $|C_\ell^{(i)}| = \sum_{\{X:\mathsf{size}(X)=i\}} |E_X| \geq (i - 3)r_i$. Thus, $r_i \leq \frac{|C_\ell^{(i)}|}{i-3}$. □

Now we can bound the cardinality of B_ℓ. Let $C_\ell^{(\leq 4)} = \bigcup_{i \leq 4} C_\ell^{(i)}$ and $C_\ell^{(\geq 5)} = \bigcup_{i \geq 5} C_\ell^{(i)}$.

$$
\begin{aligned}
|B_\ell| = \sum_i |B_\ell^{(i)}| &= \sum_i i \cdot r_i && (5.1)\\
&\leq 2|C_\ell^{(\leq 4)}| + \sum_{i \geq 5} \frac{i}{i-3} |C_\ell^{(i)}| \qquad \text{(By Claim 5.1)}\\
&\leq 2|C_\ell^{(\leq 4)}| + \frac{5}{2}|C_\ell^{(\geq 5)}| && (5.2)
\end{aligned}
$$

What remains is to bound the cardinality of B_n. Let $\mathcal{B}_n^{(\geq 3)}$ be the set of blocks in $\mathcal{B}_n$ such

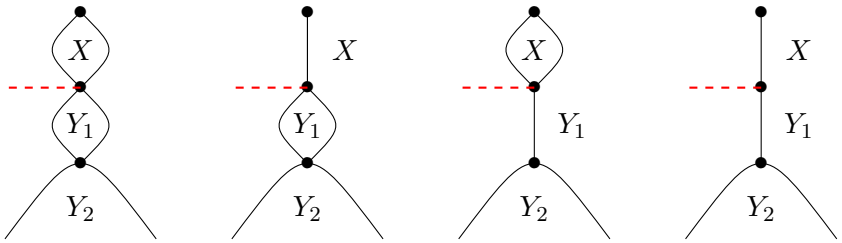


Figure 5.2: A schematic diagram, when a block X of size at most 2 has only one child which is a super block composed of Y_1 and Y_2. Here the red dotted edges belong to $E(S, V(G) \setminus S)$.

that the corresponding nodes in $\mathcal{T}$ have degree at least 3. That is,

$$\mathcal{B}_n^{(\geq 3)} = \{X \in \mathcal{B}_n \mid d_{\mathcal{T}}(t_X) \geq 3\}.$$

Let $B_n^{(\geq 3)}$ be the set of edges present in the blocks in $\mathcal{B}_n^{(\geq 3)}$. We first bound the cardinality of $B_n^{(\geq 3)}$ and then the cardinality of $B_n \setminus B_n^{(\geq 3)}$. For a set $X \subseteq V(G')$, let numcut_X and $\mathsf{numnoncut}_X$ denote the number of cut vertices and non-cut vertices in X, respectively.

$$\begin{aligned} |B_n^{(\geq 3)}| &\leq \sum_{X \in \mathcal{B}_n^{(\geq 3)}} |X| \\ &= \sum_{X \in \mathcal{B}_n^{(\geq 3)}} \mathsf{numcut}_X + \mathsf{numnoncut}_X \end{aligned} \tag{5.3}$$

The first inequality follows from the fact that the number of edges in a block of a cactus graph is at most the number of vertices in the block. The quantity $\sum_{X \in \mathcal{B}_n^{(\geq 3)}} \mathsf{numcut}_X$, is at most $\sum_{X \in \mathcal{B}_n^{(\geq 3)}} d_{\mathcal{T}}(t_X)$. This is bounded by three times the number of leaves in $\mathcal{T}$ (by Lemma 5.1). Thus by Claim 5.1,

$$\sum_{X \in \mathcal{B}_n^{(\geq 3)}} \mathsf{numcut}_X \leq \frac{3}{2}|C_\ell^{(\leq 4)}| + \frac{3}{2}|C_\ell^{(\geq 5)}| \tag{5.4}$$

Let $C_n^{\geq 3}$ be the set of edges in C_n which are incident to blocks in $\mathcal{B}_n^{(\geq 3)}$, and $C_n^{\leq 2}$ be the set of edges in C_n which are incident to blocks in $\mathcal{B}_n \setminus \mathcal{B}_n^{(\geq 3)}$. For each non-cut vertex x in the block $X \in \mathcal{B}_n^{(\geq 3)}$, there is at least one edge from $C_n^{(\geq 3)}$ which is incident on x. This implies that

$$\sum_{X \in \mathcal{B}_n^{(\geq 3)}} \mathsf{numnoncut}_X \leq |C_n^{(\geq 3)}| \tag{5.5}$$

Applying Equations 5.4 and 5.5 in Equation 5.3, we get that

$$|B_n^{(\geq 3)}| \leq \frac{3}{2}|C_\ell^{(\leq 4)}| + \frac{3}{2}|C_\ell^{(\geq 5)}| + |C_n^{(\geq 3)}| \tag{5.6}$$

We bound the cardinality of $B_n \setminus B_n^{(\geq 3)}$. First, we bound the number of edges in the blocks in $\mathcal{B}_n \setminus \mathcal{B}_n^{(\geq 3)}$ which are not incident to any edge in C_n. Let X be a block in $\mathcal{B}_n \setminus \mathcal{B}_n^{(\geq 3)}$, such that there is no edge from C_n incident on it. Since t_X has degree 2 in $\mathcal{T}$, the number of cut vertices in X is 2. We claim that $\mathsf{size}(X) \leq 2$. Suppose not. Then there is a vertex

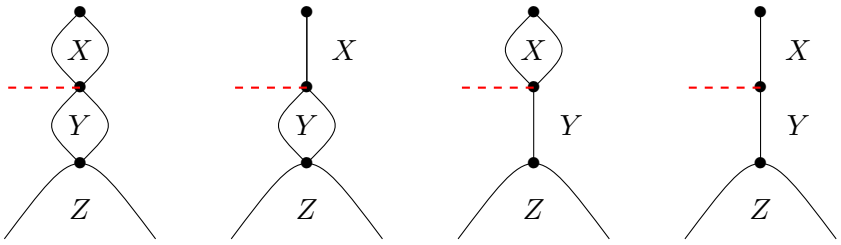


Figure 5.3: A schematic diagram, when a block X of size at most 2 has only one child Y such that $size(Y) \leq 2$ and $d_{\mathcal{T}}(t_Y) = 2$. Here the red dotted edges belong to $E(S, V(G)\backslash S)$.

x in X such that the degree of x in G' is two. Thus, by our assumption, x is incident to an edge from C_n. This contradicts the fact that there is no edge from C_n that is incident on X. Since X is a block in $\mathcal{B}_n \setminus \mathcal{B}_n^{(\geq 3)}$, we know that t_X has only one child. Let the child of t_X be t_Y. Now we have the following claim.

Claim 5.2. *Either $d_{\mathcal{T}}(t_Y) \geq 3$ or $Y \in \mathcal{B}_n \setminus \mathcal{B}_n^{(\leq 3)}$ such that there is an edge from $C_n^{(\leq 2)}$ incident on Y.*

Proof. We first show that $Y \notin \mathcal{B}_\ell$. Suppose not. If Y is a normal block in $\mathcal{B}_\ell$, then X and Y together will form a super block and it contradicts the fact that $X \in \mathcal{B}_n \setminus \mathcal{B}_n^{(\geq 3)}$. Suppose Y is a super block in $\mathcal{B}_\ell$. Let Y be the block consisting of blocks Y_1 and Y_2 where t_{Y_2} is a leaf in $\mathcal{T}$ (See Figure 5.2). Consider the shared vertex x by the blocks X and Y_1. The number of neighbours of x in G' is 2. Thus, by our assumption, x is incident with a vertex in C_n. This contradicts the fact that X be a block in $\mathcal{B}_n \setminus \mathcal{B}_n^{(\geq 3)}$ which is not incident to any edge in C_n. To prove the claim, the only case remaining is $Y \in \mathcal{B}_n \setminus \mathcal{B}_n^{(\geq 3)}$, but $d_{\mathcal{T}}(t_Y) = 2$ and there is no edge from $C_n^{(\leq 2)}$ incident on Y (See Figure 5.3). Then, the size of Y is at most 2. Consider the vertex x shared by the blocks X and Y. The number of neighbours of x in G' is 2. Thus, by our assumption, x is incident with a vertex in C_n. This contradicts the fact that X be a block in $\mathcal{B}_n \setminus \mathcal{B}_n^{(\geq 3)}$ which is not incident to any edge in C_n. This proves the claim. □

Using the above claim we can show that the total number of edges in the blocks in $\mathcal{B}_n \setminus \mathcal{B}_n^{(\geq 3)}$ which are not incident to any edge in C_n is bounded by

$$\begin{aligned} 2\left(|\mathcal{C}_n^{(\leq 2)}| + \sum_{\substack{\{t \in V(\mathcal{T}):\\ d_{\mathcal{T}}(t) \geq 3\}}} 1\right) &\leq 2|\mathcal{C}_n^{(\leq 2)}| + 2\sum_i r_i \\ &\leq 2|\mathcal{C}_n^{(\leq 2)}| + |C_\ell^{(\leq 4)}| + |C_\ell^{(\geq 5)}| \quad \text{(By Claim 5.1)} \end{aligned} \tag{5.7}$$

Next, we bound the number of edges in the blocks in $\mathcal{B}_n \setminus \mathcal{B}_n^{(\geq 3)}$ which are incident to some edges in C_n. Let X be a such a block. If the size of X is at most two, then there is at least one edge from $\mathcal{C}_n^{(\leq 2)}$ which is incident on X. If the size of X is at least $i \geq 3$, then there are $i-2$ vertices in X such that each of these vertices will have only two neighbours in G'. By our assumption, this implies that there are at least $i-2$ edges from $\mathcal{C}_n^{(\leq 2)}$ which are incident on X. Thus, the total number of edges, in the blocks in $\mathcal{B}_n \setminus \mathcal{B}_n^{(\geq 3)}$, which are incident to some edges in C_n, is bounded by $3|\mathcal{C}_n^{(\leq 2)}|$. Hence,

$$|B_n \setminus B_n^{(\geq 3)}| = 5|\mathcal{C}_n^{(\leq 2)}| + |C_\ell^{(\leq 4)}| + |C_\ell^{(\geq 5)}| \tag{5.8}$$

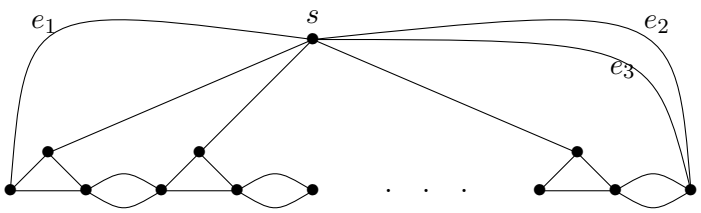


Figure 5.4: A tight example of Lemma 5.2. Here $S = \{s\}$.

Hence,

$$\begin{aligned} |B| &= |B_\ell| + |B_n^{(\geq 3)}| + |B_n \setminus B_n^{(\geq 3)}| \\ &= \frac{9}{2}|C_\ell^{(\leq 4)}| + 5|C_\ell^{(\geq 5)}| + 5|\mathcal{C}_n^{(\leq 2)}| + |C_n^{(\geq 3)}| \qquad \text{(By Equations 5.2,5.6 and 5.8)} \\ &\leq 5|C| \end{aligned}$$

This completes the proof of the Lemma. □

The bound given in Lemma 5.2 is in fact tight. Given a graph G and a set $S \subseteq V(G)$ such that the assumptions of Lemma 5.2 holdm consider the edge sets $B = E(G - S)$ and $C = E(S, V(G) \setminus S)$. Figure 5.4 represents a family of tight instances where for every pair of consecutively occurring triangle and double parallel edges in the cactus, there is an edge in C. On the other hand, except for 3 edges in C, each of the other edges are incident to one vertex of a distinct triangle. Thus, $|B| = 5(|C| - 3)$. Hence, this is a family of tight instances.

5.4 Algorithm for Even Cycle Transversal

In this section, we give a randomized FPT algorithm for Even Cycle Transversal. This problem is a special case of the following problem.

Parity Even Cycle Transversal **Parameter:** k
Input: A graph G, a weight function $w : E(G) \to \{0, 1\}$ and positive integer k
Question: Is there a set $S \subseteq V(G)$ of size at most k such that $G - S$ does not contain any cycle C with $\Sigma_{e \in E(C)} w(e) = 0 \mod 2$?

We call a cycle C an even-parity (odd-parity) cycle if $\Sigma_{e \in E(C)} w(e) = 0 \mod 2$ ($\Sigma_{e \in E(C} w(e) = 1 \mod 2$). For compactness of notation, we define the function $\mathsf{parity} : 2^{E(G)} \to \{0, 1\}$, where for an edge set $E' \in E(G)$, $\mathsf{parity}(E') = \Sigma_{e \in E'} w(e) \mod 2$. In other words, for an even-parity (odd-parity) cycle C, $\mathsf{parity}(E(C)) = 0$ ($\mathsf{parity}(E(C)) = 1$). This should not be confused with cycles of even (odd) length, since we will refer to these cycles simply as even and odd cycles.

In what follows, we give a randomized FPT algorithm for Parity Even Cycle Transversal, that runs in $12^k n^{\mathcal{O}(1)}$ time. Our algorithm will compute a vertex subset X of size at most k and return it as a solution if it is indeed a solution and otherwise return NO. First, we apply some reduction rules the input graph. A reduction rule reduces an instance (I_1, k) of a problem Π to another instance (I_2, k') of Π. The reduction rule is *safe* when

(I_1, k) is a YES instance if and only if (I_2, k') is a YES instance. Applying a reduction rule on an input graph is also termed as reducing the graph, and the resultant graph is termed as the reduced graph.

Let G be the input graph. Our algorithm will set $X := \emptyset$ initially. After the reduction rules have been exhaustively applied on the input graph G, we show that for every solution at least $\frac{1}{6}$ fraction of edges is incident with the vertices of the solution. Let G' be the reduced graph. Then our algorithm picks an edge and its endpoint (say v) at random, puts the vertex into X. Then again we apply reduction rules exhaustively on $G' - \{v\}$ such that in the reduced graph for every solution at least $\frac{1}{6}$ fraction of edges is incident with the vertices of the solution. Again our algorithm picks an edge and its endpoint at random, puts the vertex into X. The algorithm continues the above process (i.e, applying reduction rules on the graph, randomly picking an edge and choosing one of its end points) k times, or until the reduced graph is empty. If there is a solution of size at most k in G, then this procedure outputs a solution (that is, X is indeed a solution) with probability at least 12^{-k}. Then by repeating this procedure 12^k times, we obtain constant success probability.

We describe the reduction rules below and prove their safeness.

Reduction Rule 5.1. *If there is a vertex v in G which is not part of any even-parity cycle, then delete v from G.*

Lemma 5.3. *Reduction Rule 5.1 is safe.*

Proof. Suppose we delete v from G. If C is an even-parity cycle of G, it is still an even-parity cycle of $G - \{v\}$. Similarly, if there is an even-parity cycle C' in $G - \{v\}$, then C' is also a cycle in G. Now, Suppose (G, k) is a YES instance of Parity Even Cycle Transversal. Let S be a solution of size k for G. Since $G - \{v\}$ is a subgraph G and S is a solution for G, we have that $S \setminus \{v\}$ is a solution for the reduced graph $G - \{v\}$ as well. Therefore, $(G - \{v\}, k)$ is also a YES instance of Parity Even Cycle Transversal.

On the other hand, suppose the reduced instance is a YES instance. Suppose S' is a solution for $G - \{v\}$. Then, S' hits all even-parity cycles of $G - \{v\}$. This means, that S' also hits all even-parity cycles of G, and therefore S' is a solution in G. Thus, (G, k) is a YES instance of Parity Even Cycle Transversal. □

In the following Lemma, we show that, on a graph where all edges have weight 1, testing whether a vertex is contained in an even cycle can be done in polynomial time.

Lemma 5.4. *There is a polynomial time algorithm that, given a graph G, where every edge has weight 1, and a vertex $v \in V(G)$, runs in polynomial time and checks whether there is an even cycle containing v.*

Proof. For each $u \in N_G(v)$, we check whether there is an even cycle containing the edge (u, v). This is equivalent to checking whether there is an odd path P between v and w in the graph $G' = G - (u, v)$. In [Lokshtanov 2012], the Parity Multiway Cut (PMWC) problem was posed: If we are given a graph with a set of terminal vertices $T_o \uplus T_e$, does there exist a set S of at most k vertices such that $G - S$ does not have any even path between vertices of T_e and odd paths between vertices of T_o. It was shown that this problem has an FPT algorithm, when parameterized by the size k of the deletion set S.

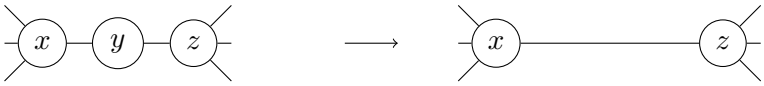


Figure 5.5: Reduction Rule 5.2. Here, $w((x,z)) = (w((x,y)) + w((y,z)) \mod 2$.

The running time of the algorithm is $2^{2^{\mathcal{O}(k)}} n^{\mathcal{O}(1)}$. We observe that our problem is a special case of the above problem. In our case, $T_o = \{u, v\}, T_e = \emptyset$ and $k = 0$. In other words, we wish to check whether there are any odd paths between u, v in G'. Since $2^{2^{\mathcal{O}(k)}} = \mathcal{O}(1)$, the algorithm for PMWC enables us to check in polynomial time, whether there are no odd paths between u and v in G'. If the algorithm returns YES, then we know that there are no even cycles in G containing the edge (u, v). Otherwise, we conclude that there is an even cycle in G containing v. If, for every edge $e \in E(G)$ adjacent to v, there is no even cycle containing the edge e, then we conclude that there is no even cycle in G containing v. □

This also gives us a polynomial time algorithm to check whether a vertex of a $(0, 1)$ edge-weighted graph is contained in an even-parity cycle.

Lemma 5.5. *There is a polynomial time algorithm that, given a graph G, where every edge has weight 0 or 1, and a vertex $v \in V(G)$, checks whether there is an even-parity cycle containing v.*

Proof. We construct, from the given graph G with an edge-weight function w, a graph $\hat{G}$ where each edge has weight 1. This is done by subdividing every edge of weight 0, and giving each of the two new edges weight 1. We mark the original vertices of G, to distinguish them from the newly introduced vertices. By this reduction, any original vertex belongs to an even-parity cycle in G if and only if it belongs to an even cycle in $\hat{G}$. Thus it is enough to check if $v \in V(G)$ belongs to an even cycle in $\hat{G}$. This can be done in polynomial time by Lemma 5.4. □

Reduction Rule 5.2. *Let $[x, y, z]$ be a path in G and degree of y is exactly 2. Then delete y from G and add a new edge $e_1 = (x, z)$ with weight $w(e_1) = w((x, y)) + w((y, z))$ mod 2. (See Figure 5.5).*

Lemma 5.6. *Reduction Rule 5.2 is safe.*

Proof. Suppose C is a cycle of parity p in G, which contains the vertex y. Then, since $d_G(y) = 2$, C must contain the path $[x, y, z]$. In the reduced graph G', C is reduced to a cycle C' which contains the edge $e_1 = (x, z)$. By definition of $w(e_1)$, the parity of the reduced cycle is still p. On the other hand, if C' is a cycle of parity p in the reduced graph G', and C' does not contain the new edge e_1, then C' is a cycle of the original graph G. Otherwise, there is a corresponding cycle C in G, which contains the path $[x, y, z]$ instead of the newly added edge e_1. Again, by definition of $w(e_1)$, the parity of C' and C are the same.

Now, suppose (G, k) is a YES instance for Parity Even Cycle Transversal. Let S be a solution set in G. Then S hits all even-parity cycles of G. We have argued that any cycle in G that contains y also contains x and z. Thus, if y was contained in S, then $(S \cup \{x\}) \setminus y$ is also a solution that hits all even-parity cycles of G. Since the parity of

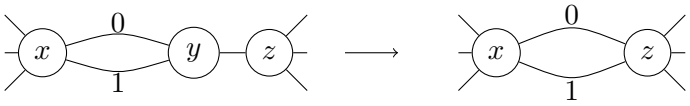


Figure 5.6: Reduction Rule 5.3.

cycles is preserved by this reduction, it implies that $(S \cup \{x\}) \setminus y$ is a solution that hits all even-parity cycles of the reduced graph, and that the reduced instance is also a YES instance.

On the other hand, suppose the reduced instance is a YES instance. let S' be a solution set of G'. We will show that S' is also a solution for G. Suppose there is an even-parity cycle C in G, that is not hit by S', then this cycle must have the vertex y. This implies that the cycle must have the path $[x, y, z]$. Let $P = C - \{y\}$. Look at the cycle $C' = P \cup e_1$ in G'. This is also an even-parity cycle which is not hit by S'. This contradicts the fact that S' is a solution set of G'. Thus, (G, k) must be a YES instance of Parity Even Cycle Transversal. □

Reduction Rule 5.3. *Let x, y be two vertices with two parallel edges e_1 and e_2. Let $w(e_1) = 1, w(e_2) = 0$. Further, $e_3 = (y, z)$ is an edge in G, with $z \neq x$, and $d_G(y) = 3$. Then delete y from the graph G and add two new edges $f_1, f_0 = (x, z)$. Define $w(f_1) = 1$ and $w(f_0) = 0$ (See Figure 5.6).*

Lemma 5.7. *Reduction Rule 5.3 is safe.*

Proof. Let G' be the reduced graph. By degree constraints on y, any even-parity cycle C containing y must also contain a path $[x, y, z]$. We give a bijective mapping Γ between the even-parity cycles of G and the even-parity cycles of G'. If C does not contain y, then this cycle exists in G' as well and $\Gamma(C) = C$ to itself. Otherwise, C contains either the path $[e_1, e_3]$ or the path $[e_2, e_3]$. Without loss of generality let the path be $[e_1, e_3]$ and let $\mathsf{parity}(e_1, e_3) = p$. Let $P = C - \{y\}$. Then G' has a cycle $C' = P \cup (f_p)$. Then, $\Gamma(C) = C'$. This mapping is parity preserving. In the reverse direction, consider an even-parity cycle C' of G'. If it does not contain one of the two edges f_1, f_0, then C' is a cycle in G and $\Gamma^{-1}(C') = C'$. Otherwise, without loss of generality, let $f_1 \in E(C)$. Let $P' = C' - f_1$. Let $e_i, i \in \{1, 2\}$ be the edge such that $w(e_i) + w(e_3) = 1$. Define $C = P' \cup \{e_i, e_3\}$. This is the only even-parity cycle in $\Gamma^{-1}(C')$. Thus Γ is bijective.

Now, suppose (G, k) is a YES instance. Let S be a solution set in G. If S contains y, then the set $(S \cup \{x\}) \setminus \{y\}$ is also a solution set in G. So, we assume that the solution set of G does not contain y. Suppose C' is an even-parity cycle in G' that is not hit by S. Then, it must be the case that $\Gamma^{-1}(C')$ contains y, and therefore a path $[e_i, (y, z)]$, $i \in \{1, 2\}$. Since we assume S to not contain y, S does not hit $\Gamma^{-1}(C')$ as well. This contradicts the fact that S was a solution set in G. Thus, the reduced instance (G', k) must be a YES instance as well.

Similarly, suppose (G', k) is a YES instance. Let S' be a solution set of G'. We will show that S' is also a solution for G. If C is an even-parity cycle of G that is not hit by S', then C must contain y. Then it must be the case that C had a path $[e_i, (y, z)]$, $i \in \{1, 2\}$, and $C' = \Gamma(C)$ has a corresponding newly added edge $f_{w(e_i)+w((y,z)) \mod 2}$. If C is not hit by

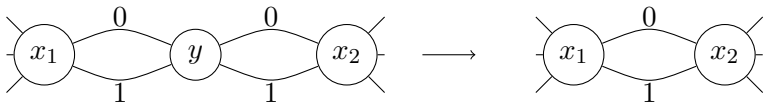


Figure 5.7: Reduction Rule 5.4.

S', then C' is also not hit. This is a contradiction that S' is a solution set for G'. Thus, the original instance must be a YES instance. □

Reduction Rule 5.4. *Let $\{x_1, y\}$ be a pair of vertices that have two parallel edges e_1 and e_2, with $w(e_1) = 1, w(e_2) = 0$. Let there be another vertex $x_2 \neq x_1$ such that $\{x_2, y\}$ have two parallel edges e_3 and e_4. It also holds that $w(e_3) = 1, w(e_4) = 0$. Let $d_G(y) = 4$. Then delete y from G and add two new parallel edges f_1, f_0 between x_1 and x_2. We define $w(f_1) = 1$ and $w(f_0) = 0$. (See Figure 5.7).*

Lemma 5.8. *Reduction Rule 5.4 is safe.*

Proof. Let G' be the reduced graph. By the degree constraint on y, any even-parity cycle C containing y must also contain a path $[e_i, e_j]$, $i \in \{1, 2\}$ and $j \in \{3, 4\}$. We give a surjective mapping Γ between the even-parity cycles of G and the even-parity cycles of G'. If C does not contain y, then this cycle exists in G' as well and $\Gamma(C) = C$ to itself. Otherwise, C contains a path $[e_i, e_j]$, $i \in \{1, 2\}$ and $j \in \{3, 4\}$. Let $P = C - \{y\}$, and let $w(e_i) + w(e_j) = p \mod 2$. Then G' has a cycle $C' = P \cup f_{p \mod 2}$. Then, $\Gamma(C) = C'$. This mapping is parity preserving. Moreover, consider an even-parity cycle C' of G'. If it does not contain one of the two edges f_1, f_0, then C' is a cycle in G and $\Gamma^{-1}(C') = C'$. Otherwise, without loss of generality, let $f_1 \in E(C)$. Let $P' = C' - f_1$. Let $e_i, e_j, i \in \{1, 2\}, j \in \{3, 4\}$, be edges such that $w(e_i) + w(e_j) = 1$. Define $C = P' \cup \{e_i, e_j\}$. This is an even-parity cycle in $\Gamma^{-1}(C')$. Thus Γ is surjective.

Now, suppose (G, k) is a YES instance. Let S be a solution set in G. If S contains y, then the set $(S \cup \{x\}) \setminus \{y\}$ is also a solution set in G. So, we assume that the solution set of G does not contain y. Suppose C' is an even-parity cycle in G' that is not hit by S. Then the cycle C' must contain the vertices x_1, x_2. Then, it must be the case that any cycle in $\Gamma^{-1}(C')$ contains y, and therefore a path $[e_i, e_j]$, $i \in \{1, 2\}$ and $j \in \{3, 4\}$. Since we assume S to not contain y, S does not hit cycles in $\Gamma^{-1}(C')$ as well. This contradicts the fact that S was a solution set in G. Thus, (G', k) must be a YES instance.

Similarly, suppose (G', k) is a YES instance. Let S' be a solution set of G'. We will show that S' is also a solution for G. If C is an even-parity cycle of G that is not hit by S', it must contain y. Then it must be the case that C had a path $[e_i, e_j]$, $i \in \{1, 2\}$ and $j \in \{3, 4\}$, and $C' = \Gamma(C)$ has at least one of the newly added edges f_1, f_2. If C is not hit by S', then C' is also not hit. This is a contradiction that S' is a solution set for G'. This means that the original instance must be a YES instance. □

We give the definition of an odd-parity (even-parity) cactus graph and relate it to Parity Even Cycle Transversal.

Definition 5.4. *A cactus graph, where the edges have weights from $\{0, 1\}$, is an odd-parity (even-parity) cactus graph when every block of the graph is either an odd-parity (even-parity) cycle or an edge.*

Lemma 5.9. *Let G be a connected graph and $w : E(G) \to \{0, 1\}$ be a weight function on the edges. The graph G does not contain any cycle C with $w(C) = 0 \mod 2$ if and only if G is an odd-parity cactus graph.*

Proof. Suppose G does not contain any even-parity cycle. Then every cycle in G must be of odd-parity. Thus, if G was a cactus graph then it must be an odd-parity cactus graph. Suppose G is not a cactus graph. Then, by Proposition 5.1, there is a diamond D in G. Let the diamond be defined at the vertex pair $\{u, v\}$ by the three disjoint paths P_1, P_2, P_3. Let $\mathsf{parity}(P_1) = p_1, \mathsf{parity}(P_2) = p_2, \mathsf{parity}(P_3) = p_3$. By Pigeonhole Principle, at least two among P_1, P_2 and P_3 must have the same parity. Without loss of generality, let P_1 and P_2 have the same parity. Then the cycle $[v, P_1, u, P_2, v]$ is of even parity, which is a contradiction. Hence, G must be an odd-parity cactus graph.

On the other hand, suppose G is an odd-parity cactus graph. Then there is a block decomposition of G where every block is either an odd-parity cycle or an edge. By definition of a block, any cycle C of G must be contained completely inside a block. This implies that there are no even-parity cycles in G. □

Given a graph G, let S be a set of vertices that hits all even-parity cycles. Then each component of $G - S$ does not contain an even-parity cycle. By Lemma 5.9, it follows that $G - S$ is a forest of odd-parity cacti.

Observation 5.1. *Let G be a reduced graph for* Parity Even Cycle Transversal *and S be a solution for* Parity Even Cycle Transversal *in G. Then, for each connected component C of $G - S$, $G[V(C) \cup S]$ and S satisfy the conditions of Lemma 5.2.*

Proof. Let $v \in C$ be a vertex that does not have at least three distinct neighbours in G. Suppose there is at most one edge in $E(v, S)$. Also note that v cannot have one neighbour in $V(C)$ with at least two parallel edges of the same parity: this would mean that two parallel edges of the same parity form an even-parity cycle. Also, notice that if v has one neighbour with at least three parallel edges, then by pigeonhole principle, at least two of the parallel edges are of the same parity. Since Reduction Rule 5.1 does not apply any more, v must have exactly two distinct neighbours. Since Reduction Rules 5.2, 5.3 and 5.4 are no longer applicable, a vertex with exactly two distinct neighbours does not exist in the reduced graph. This is a contradiction. Thus, in the reduced instance, every vertex in C satisfies the conditions of Lemma 5.2. □

Now, we are ready to describe the algorithm for Parity Even Cycle Transversal. Informally, the algorithm runs for 12^k rounds. In each round, a vertex subset of size at most k is obtained. We show that, given a YES instance, with high probability there is at least one round where the constructed vertex subset is a solution set for Parity Even Cycle Transversal. A NO instance is always detected correctly by the algorithm.

Theorem 5.2. Parity Even Cycle Transversal *has a randomized algorithm with worst case running time $12^k n^{\mathcal{O}(1)}$, where n and m are the number of vertices and edges in the input graph, respectively. The algorithm outputs* NO *if the input is a* NO *instance and for a* YES *instance, with probability $1 - \frac{1}{e}$, returns a solution.*

Proof. Let (G, k) be the input instance. Our algorithm runs a procedure (call it procedure Q) 12^k times. The procedure Q has at most k iterative steps and is as follows: We set

$S := \emptyset$ and $G' := G$ to start with. We apply Reduction Rules 5.1, 5.2, 5.3 and 5.4 to the graph G' as long as we can. If the reduced graph G'' is non-empty, we pick an edge $e = (u, v) \in E(G'')$ uniformly at random and then, with equal probability, we pick one of the two endpoints (say the vertex picked is v). In other words, we pick a vertex with probability proportional to its degree. Now we set $S := S \cup \{v\}$ and $G' := G'' - \{v\}$. We do this for at most k steps, stopping whenever the graph becomes empty. Notice that the algorithm could stop if the graph becomes empty after applying the reduction rules exhaustively. Then we check if the constructed set S is a solution set of Parity Even Cycle Transversal for the input graph G. Note that recognizing a forest of odd-parity cacti is equivalent to building a block-decomposition and checking if each block is a odd-parity cycle or an edge. If all the 12^k executions of procedure Q fail to find out a solution, then the algorithm will output NO.

Now we analyse the success probability of the algorithm. For any $i \in \{0, \ldots, k\}$, let S_i be the set of vertices obtained at the end of step i. Consider the step $i + 1$, where $i \in \{0, \ldots, k - 1\}$. Let G_{i+1} be the reduced graph in step $i + 1$. By the correctness of the reduction rules, if D is solution of cardinality at most $k - i$ for $(G_{i+1}, k - i)$, then $S_i \cup D$ is a solution for (G, k). Suppose there is a solution S^*_{k-i} of size at most $k - i$ in G_{i+1}. By Observation 5.1, for each component C of $G_{i+1} - S^*_{k-i}$, $G_{i+1}[V(C) \cup S^*_{k-i}]$ and S^*_{k-i} satisfy the conditions of Lemma 5.2. By the conditions of Lemma 5.2, for each component C of $G_{i+1} - S^*_{k-i}$, $|E(C)| \leq \frac{5}{6}|E(G_{i+1}[V(C) \cup S^*_{k-i}])|$. This implies that $|E(G_{i+1} - S^*_{k-i})| \leq \frac{5}{6}|E(G_{i+1})|$. The algorithm chooses a vertex in step $i + 1$ using a random process. We say that the vertex chosen by the algorithm in step $i + 1$ is good if the algorithm chooses a vertex from S^*_{k-i}. Since $|E(G_{i+1} - S^*_{k-i})| \leq \frac{5}{6}|E(G_{i+1})|$, the probability that an edge incident with a vertex from S^*_{k-i}, is picked uniformly at random in step $i + 1$, is at least $\frac{1}{6}$. Once we have picked this edge, the probability that we choose an end point of the edge that belongs to S^*_{k-i} is at least $\frac{1}{2}$. Therefore, the probability that a good vertex is chosen in step $i + 1$ is at least $\frac{1}{2} \cdot \frac{1}{6} = \frac{1}{12}$. We succeed in finding a solution set S for Parity Even Cycle Transversal if every step picks a good vertex in that step. Thus, the probability of failure in the k-step procedure is at most $1 - (\frac{1}{12})^k$. We repeat the procedure Q 12^k times. The probability of failure of this many-round procedure is the probability that procedure Q fails in all the 12^k executions, which is at most $(1 - (\frac{1}{12})^k)^{12^k} \leq \frac{1}{e}$.

Now we prove the claimed running time. By Lemma 5.5 we can identify and apply Reduction Rule 5.1 in polynomial time. Notice that checking whether any of Reduction Rules 5.2, 5.3 and 5.4, is applicable, takes polynomial time and these reduction rules can be applied in constant time. Since each application of a reduction rule reduces the number of vertices by at least one, the total number of times these reduction rules are applicable in the procedure Q is at most n. Thus, the total time spent for applying Reduction Rules in the procedure Q is polynomial. Moreover, in each iteration of procedure Q, we pick an edge and one of its endpoints in $\mathcal{O}(m)$ time. Therefore, over k iterations, we spend $\mathcal{O}(km)$ time picking edges and a corresponding endpoint. This means that one execution of procedure Q takes polynomial time. There are 12^k executions which makes the total running time to be $12^k n^{\mathcal{O}(1)}$. □

Corollary 5.1. Even Cycle Transversal *has a randomized algorithm with worst case running time* $12^k n^{\mathcal{O}(1)}$*, where* n *and* m *are the number of vertices and edges in the input graph, respectively. The algorithm outputs* NO *if the input is a* NO *instance and for a* YES *instance, with probability* $1 - \frac{1}{e}$*, returns a solution.*

5.5 Algorithm for Diamond Hitting Set

In this section, we give a randomized FPT algorithm for Diamond Hitting Set. It was shown in [Fiorini 2010] that there is a set of safe reduction rules that can be applied to reduce the input graph to a graph with certain properties.

Proposition 5.2 ([Fiorini 2010])**.** *There are polynomial time reduction rules, on application of which, the input instance of* Diamond Hitting Set *is reduced to an equivalent instance where every vertex either has at least three distinct neighbours or three parallel edges.*

Observation 5.2. *Let G be a reduced graph for* Diamond Hitting Set *and S be a solution in G. Then, for each connected component C in $G - S$, $G[V(C) \cup S]$ and S satisfy the conditions of Lemma 5.2.*

Proof. Let G be the reduced instance. Given a diamond-hitting set S, Proposition 5.1 shows that $G - S$ must be a forest of cacti. Thus, for each component C of $G - S$, C is a cactus graph. Let $v \in C$ be a vertex that does not have at least three distinct neighbours. Then, v must have at least three parallel edges with a neighbour u. Since there are no diamonds in C, it must be the case that $u \in S$ and therefore, there are at least two edges in $E(v, S)$. Thus, in the reduced instance, every vertex in C satisfies the conditions of Lemma 5.2. □

Now, we can design an algorithm for Diamond Hitting Set, that is very similar to the algorithm for Parity Even Cycle Transversal.

Theorem 5.3. Diamond Hitting Set *has a randomized algorithm with worst case running time $12^k n^{\mathcal{O}(1)}$, where n is the number of vertices in the input graph. The algorithm outputs* NO *if the input is a* NO *instance and for a* YES *instance, with probability $1 - \frac{1}{e}$, returns a solution.*

Proof. The algorithm is similar in description to the algorithm mentioned in the proof of Theorem 5.2. In this algorithm, instead of applying Reduction Rules 5.1, 5.2, 5.3 and 5.4, we exhaustively apply the reduction algorithm mentioned in Proposition 5.2 and check whether the constructed set is a diamond hitting set of G. The correctness of the algorithm follows from arguments similar to those given in the proof of Theorem 5.2; in the arguments we use property (iv) of Proposition 5.2 and replace Observation 5.1 with Observation 5.2. The claimed bound on the running time can be proved by using arguments similar to that used in the proof of Theorem 5.2. □

5.6 Chapter Summary

In this chapter, we saw a fast but randomized algorithm for the Even Cycle Transversal and Diamond Hitting Set problems. It would be interesting to find deterministic algorithms for these problems, without bowing up the running times by too much. Also, we showed that our running time analysis is tight. That is, there are instances where the given algorithms cannot do better. However, lower bounds on running time of algorithms for these problems are still open.

Part II

Geometric Covering

In this part, we study problems in geometric covering in the framework of parameterized complexity. In Chapter 6, we study the Gen-RBSC problem, which is a generalisation of the Set Cover problem. In particular, we study the Gen-RBSC-lines problem, under several natural parameterizations.

In Chapter 7, we continue with the study of variants of Set Cover in the geometric setting. We study the parameterized complexity of the problems Exact Cover, Unique Cover and Unique Set Cover for different geometric set systems.

Chapter 8 is on a colouring problem, called Conflict Free Colouring. The input instance is a hypergraph $H = (U, \mathcal{F})$ and a positive integer r, and the question is to determine if there is an r-colouring of U such that in each set of $\mathcal{F}$ there is an element that is uniquely coloured. We study a parameterized version of this problem. The main motivation for the study of the classical problem was due to applications in geometry. However, we manage to design FPT algorithms for finding a maximum sized subfamily $\mathcal{F}' \subseteq \mathcal{F}$ such that the hypergraph $H' = (U, \mathcal{F}')$ is r conflict-free colourable.

In Chapter 9, we consider the Rectilinear Steiner Tree problem, which has important and wide-ranged applications. We give the first deterministic subexponential exact algorithm for this problem. We are also able to obtain a subexponential time exact algorithm for the related problem of Rectilinear Steiner Arborescence.

Chapter 6

Multivariate Analysis of Geometric RBSC

6.1 Introduction

A set system consists of a universe U of n elements and a family $\mathcal{F}$ of m subsets of U. An input to a covering problem consists of a set system $(U, \mathcal{F})$ and a positive integer k, and the objective is to check whether there exists a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ of size at most k satisfying some desired properties. If $\mathcal{F}'$ is required to contain all the elements of U, then it corresponds to the classical Set Cover problem. The Set Cover problem is part of Karp's 21 NP-complete problems [Karp 1972]. This, together with its numerous variants, is one of the most well-studied problems in the area of algorithms and complexity. It is one of the central problems in all the paradigms that have been established to cope with NP-hardness, including approximation algorithms, randomized algorithms and parameterized complexity.

6.1.1 Problems Studied, Context and Framework

The goal of this paper is to study a generalisation of a variant of Set Cover, namely, the Red Blue Set Cover problem.

Red Blue Set Cover (RBSC)
Input: A universe $U = (R, B)$ where R is a set of r red elements and B is a set of b blue elements, a family $\mathcal{F}$ of ℓ subsets of U, and a positive integer k_r.
Question: Is there a subfamily $\mathcal{F}'$ of sets that covers all blue elements but at most k_r red elements?

Red Blue Set Cover was introduced in 2000 by Carr et al. [Carr 2000]. This problem is closely related to several combinatorial optimization problems such as the Group Steiner, Minimum Label Path, Minimum Monotone Satisfying Assignment and Symmetric Label Cover problems. This has also found applications in areas like fraud/anomaly detection, information retrieval and the classification problem. Red Blue Set Cover is NP-complete, following from an easy reduction from Set Cover itself.

We study the parameterized complexity, under various parameters, of a common generalisation of both Set Cover and Red Blue Set Cover, in a geometric setting.

GENERALISED RED BLUE SET COVER (GEN-RBSC)
Input: A universe $U = (R, B)$ where R is a set of r red elements and B is a set of b blue elements, a family $\mathcal{F}$ of ℓ subsets of U, and positive integers k_ℓ, k_r.
Question: Is there a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ of size at most k_ℓ that covers all blue elements but at most k_r red elements?

It is easy to see that when $k_\ell = |\mathcal{F}|$ then the problem instance is a RED BLUE SET COVER instance, while it is a SET COVER instance when $k_\ell = k, R = \emptyset, k_r = 0$.

In the parameterized setting, SET COVER, parameterized by k, is W[2]-hard [Downey 2012] and it is not expected to have an FPT algorithm. The NP-hardness reduction from SET COVER to RED BLUE SET COVER implies that RED BLUE SET COVER is W[2]-hard parameterized by the size k_ℓ of a solution subfamily. However, the hardness result was not the end of the story for the SET COVER problem in parameterized complexity. In literature, various special cases of SET COVER have been studied. A few examples are instances with sets of bounded size [Fellows 2008], sets with bounded intersection [Langerman 2005, Raman 2008], and instances where the bipartite incidence graph corresponding to the set family has bounded treewidth or excludes some graph H as a minor [Demaine 2005, Fomin 2009]. Apart from these results, there has also been extended study on different parameterizations of SET COVER. A special case of SET COVER which is central to the topic of this paper is the one where the *sets in the family correspond to some geometric object*. In the simplest geometric variant of SET COVER, called POINT LINE COVER, the elements of U are points in $\mathbb{R}^2$ and each set contains a maximal number of collinear points. This version of the problem is FPT and in fact has a polynomial kernel [Langerman 2005]. Moreover, the size of these kernels have been proved to be tight, under standard assumptions, in [Kratsch 2014a]. When we take the sets to be the space bounded by unit squares, SET COVER is W[1]-hard [Marx 2005]. On the other hand when surfaces of hyperspheres are sets then the problem is FPT [Langerman 2005]. There are several other geometric variants of SET COVER that have been studied in parameterized complexity, under the parameter k, the size of the solution subfamily. These geometric results motivate a systematic study of the parameterized complexity of geometric GEN-RBSC problems.

There is an array of natural parameters in hand for the GEN-RBSC problem. Hence, the problem promises an interesting dichotomy in parameterized complexity, under the various parameters. In this chapter, we concentrate on the GENERALISED RED BLUE SET COVER WITH LINES problem, parameterized under combinations of natural parameters.

GENERALISED RED BLUE SET COVER WITH LINES (GEN-RBSC-LINES)
Input: A universe $U = (R, B)$ where R is a set of r red points and B is a set of b blue points, a family $\mathcal{F}$ of ℓ sets of U such that each set contains a maximal set of collinear points of U, and positive integers k_ℓ, k_r.
Question: Is there a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ of size at most k_ℓ that covers all blue points but at most k_r red points?

It is safe to assume that $r \geq k_r$, and $\ell \geq k_\ell$. Since it is enough to find a minimal solution family $\mathcal{F}'$, we can also assume that $b \geq k_\ell$.

We finish this section with some related results. As mentioned earlier, the RED BLUE SET COVER problem in classical complexity is NP-complete. Interestingly, if the inci-

dence matrix, built over the sets and elements, has the consecutive ones property then the problem is in P [Dom 2008]. The problem has been studied in approximation algorithms as well [Carr 2000, Peleg 2007]. Specially, the geometric variant, where every set is the space bounded by a unit square, has a polynomial time approximation scheme (PTAS) [Chan 2015].

6.1.2 Our Contributions

Some of the results of this chapter are based on the works in [Ashok 2016]. In this chapter, we first show a complete dichotomy of the parameterized complexity of Gen-RBSC-lines. For a list of parameters, namely, k_ℓ, k_r, r, b, and ℓ, and all possible combinations of them, we show hardness or an FPT algorithm. Further, for parameterizations where an FPT algorithm exists, we either show that the problem admits a polynomial kernel or that it does not contain a polynomial kernel unless CoNP $\subseteq$ NP/poly.

To describe our results we first state a few definitions. For a set $S \subseteq U$, we denote by 2^S the family of all the subsets of S, and by U^S the family of all the subsets of U that contain S (that is, all supersets of S in U). For a collection $\mathfrak{F}$ of sets over a universe U, by $\mathsf{DownClosure}(\mathfrak{F})$ and $\mathsf{UpClosure}(\mathfrak{F})$ we mean the families $\bigcup_{S\in\mathfrak{F}} 2^S$ and $\bigcup_{S\in\mathfrak{F}} U^S$ respectively. Our first contribution is the following parameterized and kernelization dichotomy result for Gen-RBSC-lines.

Theorem 6.1. *Let* $\Gamma = \{\ell, r, b, k_\ell, k_r\}$. *Then* Gen-RBSC-lines *is* FPT *parameterized by* $\Gamma' \subseteq \Gamma$ *if and only if* $\Gamma' \notin \mathsf{DownClosure}(\{\{k_\ell, b\}, \{r\}\})$. *Furthermore,* Gen-RBSC-lines *admits a polynomial kernel parameterized by* $\Gamma' \subseteq \Gamma$ *if and only if* $\Gamma' \in \mathsf{UpClosure}(\{\{\ell\}, \{k_\ell, r\}, \{b, r\}\})$.

Essentially, the theorem says that if Gen-RBSC-lines is FPT parameterized by $\Gamma' \subseteq \Gamma$ then there exists an algorithm for Gen-RBSC-lines running in time $f(\Gamma') \cdot (|U| + |\mathcal{F}|)^{\mathcal{O}(1)}$. That is, the running time of the algorithm can depend in an arbitrary manner on the parameters present in Γ'. Equivalently, we have an algorithm running in time $f(\tau) \cdot (|U| + |\mathcal{F}|)^{\mathcal{O}(1)}$, where $\tau = \sum_{q\in\Gamma'} q$. Similarly, if the problem admits a polynomial kernel parameterized by Γ' then in polynomial time we get an equivalent instance of the problem of size $\tau^{\mathcal{O}(1)}$. On the other hand when we say that the problem does not admit polynomial kernel parameterized by Γ' then it means that there is no kernelization algorithm outputting a kernel of size $\tau^{\mathcal{O}(1)}$ unless CoNP $\subseteq$ NP/poly. A schematic diagram explaining the results proved in Theorem 6.1 can be seen in Figure 6.1. Results for a $\Gamma' \subseteq \Gamma$ which is not depicted in Figure 6.1 can be derived by checking whether Γ' is in $\mathsf{DownClosure}(\{\{k_\ell, b\}, \{r\}\})$.

Next we consider the RBSC-lines problem. Here we do not have any constraint on how many sets we pick in the solution family but we are allowed to cover at most k_r red points. This brings two main changes in Figure 6.1. For Gen-RBSC-lines we show that the problem is NP-hard even when there is a constant number of red points. However, RBSC-lines becomes FPT parameterized by r. In contrast, RBSC-lines is W[1]-hard parameterized by k_r. This leads to the following dichotomy theorem for RBSC-lines.

Theorem 6.2. *Let* $\Gamma = \{\ell, r, b, k_r\}$. *Then* RBSC-lines *is* FPT *parameterized by* $\Gamma' \subseteq \Gamma$ *if and only if* $\Gamma' \notin \{\{b\}, \{k_r\}\})$. *Furthermore,* RBSC-lines *admits polynomial kernel parameterized by* $\Gamma' \subseteq \Gamma$ *if and only if* $\Gamma' \in \mathsf{UpClosure}(\{\{\ell\}, \{b, r\}\})$.

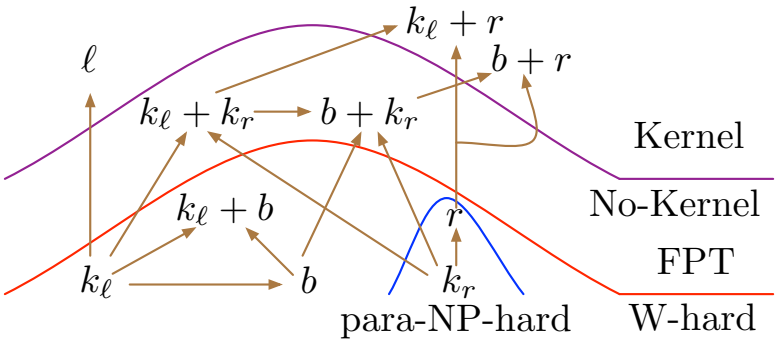


Figure 6.1: Illustration of our results described in Theorem 6.1 and hierarchy of parameters.

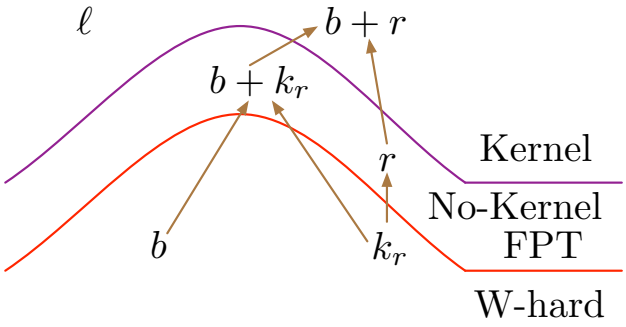


Figure 6.2: Illustration of our results for Red Blue Set Cover with lines under various parameters.

A schematic diagram explaining the results proved in Theorem 6.2 is given in Figure 6.2.

A quick look at Figure 6.1 will show that the Gen-RBSC-lines problem is FPT parameterized by $k_\ell + k_r$ or $b + k_r$. A natural question to ask is whether Gen-RBSC itself (the problem where sets in the input family are arbitrary and do not correspond to lines) is FPT when parameterized by $b + k_r$. Regarding this, we show the following results:

1. Gen-RBSC is W[1]-hard parameterized by $k_\ell + k_r$ (or $b + k_r$) when every set has size at most three and contains at least two red points.

2. Gen-RBSC is W[2]-hard parameterized by $k_\ell + r$ when every set contains at most one red point.

The first result essentially shows that Gen-RBSC is W[1]-hard even when the sets in the family have size *bounded by three*. This is in sharp contrast to Set Cover, which is known to be FPT parameterized by k_ℓ and d. Here, d is the size of the maximum cardinality set in $\mathcal{F}$. In fact, Set Cover admits a kernel of size $k_\ell^{\mathcal{O}(d)}$. This leads to the following question:

Does the hardness of Gen-RBSC in item one arise from the presence of two

red points in the instance? Would the complexity change if we assume that each set contains at most one red point?

In fact, even if we assume that each set contains at most one red point, we must take d, the size of the maximum cardinality set in $\mathcal{F}$, as a parameter. Else, this would correspond to the hardness result presented in item two. As a final algorithmic result we show that Gen-RBSC admits an algorithm with running time $2^{\mathcal{O}(dk_\ell)} \cdot (|U| + |\mathcal{F}|)^{\mathcal{O}(1)}$, when every set has at most one red point. Observe that in this setting k_r can always be assumed to be less than k_ℓ. Thus, this is also a FPT algorithm parameterized by $k_\ell + k_r$, when sets in the input family are bounded. However, we show that Gen-RBSC (in fact Gen-RBSC-lines) does not admit a polynomial kernel parameterized by $k_\ell + k_r$ even when each set in the input family corresponds to a line and has size two and contains at most one red point.

6.1.3 Our methods and an overview of main algorithmic results

Let $\Gamma = \{\ell, r, b, k_\ell, k_r\}$. Most of our W-hardness results for a Gen-RBSC variant parameterized by $\Gamma' \subseteq \Gamma$ are obtained by giving a polynomial time reduction, from Set Cover or Multicoloured Clique that makes every $q \in \Gamma'$ at most $k^{\mathcal{O}(1)}$ (in fact most of the time $\mathcal{O}(k)$). This allows us to transfer the known hardness results about Set Cover and Multicoloured Clique to our problem. Since in most cases the parameters are linear in the input parameter, in fact we can rule out an algorithm of form $(|U| + |\mathcal{F}|)^{o(\tau)}$, where $\tau = \sum_{q \in \Gamma'} q$, under Exponential Time Hypothesis (ETH) [Impagliazzo 2001]. Similarly, hardness results for kernels are derived from giving an appropriate polynomial time reduction from parameterized variants of the Set Cover problem that only allows each parameter $q \in \Gamma'$ to grow polynomially in the input parameter.

Our main algorithmic highlights are parameterized algorithms for

(a) Gen-RBSC-lines running in time $2^{\mathcal{O}(k_\ell \log k_\ell + k_r \log k_r)} \cdot (|U| + |\mathcal{F}|)^{\mathcal{O}(1)}$ (showing Gen-RBSC-lines is FPT parameterized by $k_\ell + k_r$); and

(b) Gen-RBSC with running time $2^{\mathcal{O}(dk_\ell)} \cdot (|U| + |\mathcal{F}|)^{\mathcal{O}(1)}$, when every set is of size at most d and has at most one red point.

Observe that the first algorithm generalises the known algorithm for Point Line Cover which runs in time $2^{\mathcal{O}(k_\ell \log k_\ell)} \cdot (|U| + |\mathcal{F}|)^{\mathcal{O}(1)}$ [Langerman 2005].

The parameterized algorithm for Gen-RBSC-lines mentioned in (a) starts by bounding the number of blue vertices by k_ℓ^2 and guessing the lines that contain at least two blue points. The number of lines containing at least two blue points can be shown to be at most k_ℓ^4. These guesses lead to an equivalent instance where each line contains exactly one blue point and there are no lines that only contain red points (as these lines can be deleted). However, we can not bound the number of red points at this stage. We introduce a notion of "solution subfamily" and connected components of the solution subfamilies. Interestingly, this equivalent instance has sufficient geometric structure on the connected components. We exploit the structure of these components, gotten mainly from simple properties of lines on a plane, to show that knowing one of the lines in each component can, in FPT time, lead to finding the component itself! Thus, to find a component all we

need to do is to guess one of the lines in it. However, here we face our second difficulty: the number of connected components can be as bad as $\mathcal{O}(k_\ell)$ and thus if we guess one line for each connected component then it would lead to a factor of $|\mathcal{F}|^{\mathcal{O}(k_\ell)}$ in the running time of the algorithm. However, our equivalent instances are such that we are allowed to process each component independent of other components. This brings the total running time of guessing the first line of each component down to $k_\ell \cdot |\mathcal{F}|$. The algorithmic ideas used here can be viewed as some sort of "geometry preserving subgraph isomorphism", which could be useful in other contexts also. This completes an overview of the FPT result for Gen-RBSC-lines parameterized by $k_\ell + k_r$.

The algorithm for Gen-RBSC running in time $2^{\mathcal{O}(dk_\ell)} \cdot (|U| + |\mathcal{F}|)^{\mathcal{O}(1)}$, where every set is of size at most d and has at most one red point is purely based on a novel reduction to Subgraph Isomorphism where the subgraph we are looking for has size $\mathcal{O}(k_\ell d)$ and treewidth 3. The host graph, where we are looking for a solution subgraph, is obtained by starting with the bipartite incidence graph and making modifications to it. The bipartite incidence graph we start with has in one side vertices for sets and in the other side vertices corresponding to blue and red points and there is an edge between vertices corresponding to a set and a blue (red) point if this blue (red) point is contained in the set. Our main observation is that a solution subfamily can be captured by a subgraph of size $\mathcal{O}(k_\ell d)$ and treewidth 3. Thus, for our algorithm we enumerate all such subgraphs in time $2^{\mathcal{O}(dk_\ell)} \cdot (|U| + |\mathcal{F}|)^{\mathcal{O}(1)}$ and for each such subgraph we check whether it exists in the host graph using known algorithms for Subgraph Isomorphism. This concludes the description of this algorithm.

6.2 Preliminaries

Generalised Red Blue Set Cover. A set S in a Generalised Red Blue Set Cover instance $(U, \mathcal{F})$ is said to *cover* a point $p \in U$ if $p \in S$. A *solution family* for the instance is a family of sets of size at most k_ℓ that covers all the blue points and at most k_r red points. In case of Red Blue Set Cover, the solution family is simply a family of sets that covers all the blue points but at most k_r red points. Such a family will also be referred to as a *valid family*. A *minimal family of sets* is a family of sets such that every set contains a unique blue point. In other words, deleting any set from the family implies that a strictly smaller set of blue points is covered by the remaining sets. The sets of Generalised Red Blue Set Cover with lines are also called *lines*. We also mention a key observation about lines in this section. This observation is crucial in many arguments in this paper.

Observation 6.1. *Given a set of points S, let $\mathcal{F}$ be the set of lines such that each line contains at least 2 points from S. Then $|\mathcal{F}| \leq \binom{|S|}{2}$.*

Gen-RBSC with hyperplanes of $\mathbb{R}^d$, for a fixed positive integer d, is a special case for the problem. Here, the input universe U is a set of n points in $\mathbb{R}^d$. A hyperplane in $\mathbb{R}^d$ is the affine hull of a set of $d+1$ affinely independent points [Langerman 2005]. In our special case each set is a maximal set of points that lie on a hyperplane of $\mathbb{R}^d$.

Definition 6.1. *An intersection graph $G_\mathcal{F}$ for an instance $(U, \mathcal{F})$ of* Generalised Red Blue Set Cover *is a graph with vertices in $V(G_\mathcal{F})$ corresponding to the sets in $\mathcal{F}$. We give an edge between two vertices if the corresponding sets have non-empty intersection.*

The following proposition is a collection of results on the Set Cover problem, that will be repeatedly used in the paper. The results are from [Dom 2014, Downey 2012]

Proposition 6.1. *The* Set Cover *problem is:*

1. W[2]*-hard when parameterized by the solution family size* k*.*
2. FPT *when parameterized by the universe size* n*, but does not admit polynomial kernels unless* CoNP $\subseteq$ NP/*poly.*
3. FPT *when parameterized by the number of sets* m *in the instance, but does not admit polynomial kernels unless* CoNP $\subseteq$ NP/*poly.*

6.3 Parameterizing by k_r and r

In this section, we first show that Gen-RBSC-lines parameterized by r is para-NP-complete. Since $k_r \leq r$, it follows that Gen-RBSC-lines parameterized by k_r is also para-NP-complete.

Theorem 6.3. Gen-RBSC-lines *is para-*NP*-complete parameterized by either* r *or* k_r*.*

Proof. If we are given a solution family for an instance of Gen-RBSC-lines we can check in polynomial time if it is valid. Hence, Gen-RBSC-lines has a nondeterministic algorithm with FPT running time (in fact polynomial) and thus Gen-RBSC-lines parameterized by r is in para-NP.

For completeness, there is an easy polynomial-time many-one reduction from the Point Line Cover problem, which is NP-complete. An instance $((U, \mathcal{F}))$ of Point Line Cover parameterized by k, the size of the solution family, is reduced to an instance $((R \cup B, \mathcal{F}))$ of Gen-RBSC-lines parameterized by r or k_r with the following properties:

- $B = U$
- The family of sets remains the same in both instances.
- R consists of 1 red vertex that does not belong to any of the lines of $\mathcal{F}$.
- $k_\ell = k$ and $k_r = 0$.

It is easy to see that $((U, \mathcal{F}))$ is a YES instance of Point Line Cover if and only if $(R \cup B, \mathcal{F})$ is a YES instance of Gen-RBSC-lines. Since the reduced instances belong to Gen-RBSC-lines parameterized by $r = 1$ or $k_r = 0$, this proves that Gen-RBSC-lines parameterized by r or k_r is para-NP-complete. □

6.4 Parameterizing by ℓ

In this section, we design a parameterized algorithm as well as a kernel for Gen-RBSC-lines when parameterized by the size ℓ of the family. The algorithm for this is simple.

We enumerate all possible k_ℓ-sized subsets of input lines and for each subset, we check in polynomial time whether it covers all blue points and at most k_r red points. The algorithm runs in time $\mathcal{O}(2^\ell \cdot (|U| + |\mathcal{F}|)$. The main result of this section is a polynomial kernel for Gen-RBSC-lines when parameterized by ℓ.

We start by a few reduction rules which will be used not only in the kernelization algorithm given below but also in other parameterized and kernelization algorithms in subsequent sections.

Reduction Rule 6.1. *If there is a set $S \in \mathcal{F}$ with only red points then delete S from $\mathcal{F}$.*

Lemma 6.1. *Reduction Rule 6.1 is safe.*

Proof. Let $\mathcal{F}'$ be a family of at most k_ℓ lines of the given instance that cover all blue points and at most k_r red points. If $\mathcal{F}'$ contains S, then $\mathcal{F}' \setminus \{S\}$ is also a family of at most k_ℓ lines that cover all blue points and at most k_r red points. Hence, we can safely delete S. This shows that Reduction Rule 6.1 is safe. □

Reduction Rule 6.2. *If there is a set $S \in \mathcal{F}$ with more than k_r red points in it then delete S from $\mathcal{F}$.*

Lemma 6.2. *Reduction Rule 6.2 is safe.*

Proof. If S has more than k_r red points then S alone exceeds the budget given for the permissible number of covered red points. Hence, S cannot be part of any solution family and can be safely deleted from the instance. This shows that Reduction Rule 6.2 is safe. □

Our final rule is as follows. A similar Reduction Rule was used in [Langerman 2005], for the Point Line Cover problem.

Reduction Rule 6.3. *If there is a set $S \in \mathcal{F}$ with at least $k_\ell + 1$ blue points then reduce the budget of k_ℓ by 1 and the budget of k_r by $|R \cap S|$. The new instance is $(U \setminus S, \widetilde{\mathcal{F}})$, where $\widetilde{\mathcal{F}} = \{F \setminus S \mid F \in \mathcal{F} \text{ and } F \neq S\}$.*

Lemma 6.3. *Reduction Rule 6.3 is safe.*

Proof. If S is not part of the solution family then we need at least $k_\ell + 1$ lines in the solution family to cover the blue points in S, which is not possible. Hence any solution family must contain S.

Suppose the reduced instance has a solution family $\mathcal{F}'$ covering $B \setminus S$ blue points and at most $k_r - |R \cap S|$ red points from $R \setminus S$. Then $\mathcal{F}' \cup \{S\}$ is a solution for the original instance. On the other hand, suppose the original instance has a solution family $\hat{\mathcal{F}}$. As argued above, $S \in \hat{\mathcal{F}}$. $\hat{\mathcal{F}} \setminus S$ covers all blue points of $B \setminus S$ and at most $k_r - |R \cap S|$ red points from $R \setminus S$, and is a candidate solution family for the reduced instance. Thus, Reduction Rule 6.3 is safe. □

The following simple observation can be made after exhaustive application of Reduction Rule 6.3.

Observation 6.2. *If the budget for the subfamily $\mathcal{F}'$ to cover all blue and at most k_r red points is k_ℓ then after exhaustive applications of Reduction Rule 6.3 there can be at most $b \leq k_\ell^2$ blue points remaining in a* YES *instance. If there are more than k_ℓ^2 blue points remaining to be covered then we correctly say* NO.

It is worth mentioning that even if we had weights on the red points in R and asked for a solution family of size at most k_ℓ that covered all blue points but red points of weight at most k_r, then this weighted version, called Weighted Gen-RBSC-lines parameterized by ℓ is FPT. The Weighted Gen-RBSC-lines problem will be useful in the theorem below. Finally, we get the following result.

Theorem 6.4. *There is an algorithm for* Gen-RBSC-lines *running in time $\mathcal{O}(2^\ell \cdot (|U| + |\mathcal{F}|))$. In fact,* Gen-RBSC-lines *admits a polynomial kernel parameterized by ℓ.*

Proof. We have already described the enumeration based algorithm at the beginning of this section. Here, we only give the polynomial kernel. Given an instance of Gen-RBSC-lines we exhaustively apply Reduction Rules 6.1, 6.2 and 6.3 to obtain an equivalent instance. By Observation 6.2 and the fact that $k_\ell \leq \ell$, the current instance must have at most ℓ^2 blue points, or we can safely say NO. Also, the number of red points that belong to 2 or more lines is bounded by the number of intersection points of the ℓ lines, i.e., ℓ^2. Any remaining red points belong to exactly 1 line. We reduce our Gen-RBSC-lines instance to a Weighted Gen-RBSC-lines instance as follows:

- The family of lines and the set of blue points remain the same in the reduced instance. The red points appearing in the intersection of two lines also remain the same. Give a weight of 1 to these red points.
- For each line L, let $c(L)$ indicate the number of red points that belong exclusively to L. Remove all but one of these red points and give weight $c(L)$ to the remaining exclusive red point.

In the Weighted Gen-RBSC-lines instance, there are ℓ lines, at most ℓ^2 blue points and at most $\ell^2 + \ell$ red points. For each line L, the value of $c(L)$ is at most k_r, after Reduction Rule 6.2. Suppose $k_r > 2^\ell$. Then $r > 2^\ell$ and the parameterized algorithm for Gen-RBSC-lines running in time $\mathcal{O}(2^\ell \cdot (|U| + |\mathcal{F}|))$ runs in polynomial time. Thus we can assume that $k_r \leq 2^\ell$. Then we can represent k_r and therefore the weights $c(L)$ by at most ℓ bits. Thus, the reduced instance has size bounded by $\mathcal{O}(\ell^2)$.

Observe that we got an instance of Weighted Gen-RBSC-lines and not of Gen-RBSC-lines which is the requirement for the kernelization procedure. All this shows is that the reduction is a "compression" from Gen-RBSC-lines parameterized by ℓ to Weighted Gen-RBSC-lines parameterized by ℓ. This is rectified as follows. Since both the problems belong to NP, there is a polynomial time many-one reduction from Weighted Gen-RBSC-lines to Gen-RBSC-lines. Finally, using this polynomial time reduction, we obtain a polynomial size kernel for Gen-RBSC-lines parameterized by ℓ. □

Observe that the algorithm referred to in Theorem 6.4 does not use the fact that sets are lines and thus it also works for Gen-RBSC parameterized by ℓ. However, since Set Cover is a special case of Gen-RBSC, when there are no input red vertices, it follows from Proposition 6.1(iii) that Gen-RBSC parameterized by ℓ does not admit a polynomial kernel.

6.5 Parameterizing by k_ℓ, b and $k_\ell + b$

In this section, we look at Gen-RBSC-lines parameterized by k_ℓ, b, and $k_\ell + b$. There is an interesting connection between b and k_ℓ. As we are looking for minimal solution families, we can alway assume that $b \geq k_\ell$. On the other hand, Reduction Rule 6.3 showed us that for all practical purposes $b \leq k_\ell^2$. Thus, in the realm of parameterized complexity k_ℓ, b and $k_\ell + b$ are the *same parameters*. That is, Gen-RBSC-lines is FPT parameterized by k_ℓ if and only if it is FPT parameterized by b if and only if it is FPT parameterized by $k_\ell + b$. The same holds in the context of kernelization complexity. First, we show that Gen-RBSC-lines parameterized by k_ℓ or b is W[1]-hard. Then we look at some special cases that turn out to be FPT.

6.5.1 Parameter $k_\ell + b$

We look at Gen-RBSC-lines parameterized by $k_\ell + b$. This problem is not expected to have a FPT algorithm as it is W[1]-hard. We give a reduction to this problem from the Multicoloured Clique problem, which is known to be W[1] hard even on regular graphs [Mathieson 2008].

Multicoloured Clique	**Parameter:** k
Input: A graph G where $V(G) = V_1 \uplus V_2 \uplus \ldots \uplus V_k$ with V_i being an independent set for all $1 \leq i \leq k$, and an integer k.	
Question: Is there a clique $C \leq_s G$ of size k such that $\forall 1 \leq i \leq k, V(C) \cap V_i \neq \emptyset$.	

The clique containing one vertex from each part is called a *multicoloured clique*.

Theorem 6.5. Gen-RBSC-lines *parameterized by k_ℓ or b or $k_\ell + b$ is* W[1]*-hard.*

Proof. We will give a reduction from Multicoloured Clique on regular graphs. Let (G, k) be an instance of Multicoloured Clique, where G is a d-regular graph. We construct an instance of Gen-RBSC-lines $(R \cup B, \mathcal{F})$, as follows. Let $V(G) = V_1 \uplus V_2 \uplus \ldots \uplus V_k$.

1. For each vertex class $V_i, 1 \leq i \leq k$, add two blue points b_i at $(0, i)$ and b'_i at $(i, 0)$.
2. Informally, for each vertex class $V_i, 1 \leq i \leq k$ we do as follows. Let L_k be the line that is parallel to y axis and passes through the point $(k, 0)$. Suppose there are n_i vertices in V_i. We select n_i distinct points, say $\mathcal{P}$, in $\mathbb{R}^2$ on the line L, such that if $(a_i, a_2) \in \mathcal{P}$ then $a_i = k$ (as these are points on L_k) and a_2 lies in the interval $(i - 1, i - \frac{1}{2})$. Now for every point $p \in \mathcal{P}$ we draw the unique line between $(0, i)$ and the point p. Finally, we assign each line to a unique vertex in V_i. Formally, we do as follows. For each vertex class $V_i, 1 \leq i \leq k$ and each vertex $u \in V_i$, we choose a point $p_u^1 \in \mathbb{R}^2$ with coordinates (k, y_u), $i - 1 < y_u < i - \frac{1}{2}$. Also, for each pair $u \neq v \in V_i$, $y_u \neq y_v$. For each $u \in V_i$, we add the line l_u^1, defined by b_i and p_u^1, to $\mathcal{F}$. We call these *near-horizontal lines*. Observe that all the near-horizontal lines corresponding to vertices in V_i intersect at b_i. Furthermore, for any two vertices $u \in V_i$ and $v \in V_j$, with $i \neq j$, the lines l_u^1 and l_v^1 do not intersect on a point with x-coordinate from the closed interval $[0, k]$.

3. Similarly, for each vertex class $V_i, 1 \leq i \leq k$ and each vertex $u \in V_i$, we choose a point $p_u^2 \in \mathbb{R}^2$ with coordinates $(x_u, k), i-1 < x_u < i - \frac{1}{2}$. Again, for each pair $u \neq v \in V_i$, $y_u \neq y_v$. For each $u \in V_i$, we add the line l_u^2, defined by b_i and p_u^2, to $\mathcal{F}$. Notice that for any $u, v \in V$, l_u^1 and l_v^2 have a non-empty intersection. We call these *near-vertical lines*. Observe that all the near-vertical lines corresponding to vertices in V_i intersect at b_i'. Furthermore, for any two vertices $u \in V_i$ and $v \in V_j$, with $i \neq j$, the lines l_u^2 and l_v^2 do not intersect on a point with y-coordinate from the closed interval $[0, k]$. However, a near-horizontal line and a near-vertical line will intersect at a point with both x and y-coordinate from the closed interval $[0, k]$. The construction ensures that no 3 lines in $\mathcal{F}$ have a common intersection.

4. For each edge $e = uv \in E(G)$, add two red points, r_{uv} at the intersection of lines l_u^1 and l_v^2, and r_{vu} at the intersection of lines l_v^1 and l_u^2.

5. For each vertex $v \in V(G)$, add a red point at the intersection of the lines l_v^1 and l_v^2.

This concludes the description of the reduced instance. Thus we have an instance $(R \cup B, \mathcal{F})$ of Gen-RBSC-lines with $2n$ lines, $2k$ blue points and $2m + n$ red points.

Claim 6.1. *G has a multicoloured clique of size k if and only if $(R \cup B, \mathcal{F})$ has a solution family of $2k$ lines, covering the $2k$ blue points and at most $2(d+1)k - k^2$ red points.*

Proof. Assume there exists a multicoloured clique C of size k in G. Select the $2k$ lines corresponding to the vertices in the clique. That is, select the subset of lines $\mathcal{F}' = \{l_u^j \mid 1 \leq j \leq 2, u \in V(C)\}$ in the Gen-RBSC-lines instance. Since the clique is multicoloured, these lines cover all the blue points. Each line (near-horizontal or near-vertical) has exactly $d+1$ red points. Thus, the number of red points covered by $\mathcal{F}'$ is at most $(d+1)2k$. However, each red point corresponding to vertices in $V(C)$ and the two red points corresponding to each edge in $E(C)$ are counted twice. Thus, the number of red points covered by $\mathcal{F}'$ is at most $(d+1)2k - k - 2\binom{k}{2} = 2(d+1)k - k^2$. This completes the proof in the forward direction.

Now, assume there is a minimal solution family of size at most $2k$, containing at most $2(d+1)k - k^2$ red points. As no two blue points are on the same line and there are $2k$ blue points, there exists a unique line covering each blue point. Let $\mathcal{L}^1$ and $\mathcal{L}^2$ represent the sets of near-horizontal and near-vertical lines respectively in the solution family. Observe that $\mathcal{L}^1$ covers $\{b_1, \ldots, b_k\}$ and $\mathcal{L}^2$ covers $\{b_1', \ldots, b_k'\}$. Let $V(C) = \{v_1, \ldots, v_k\}$ be the set of vertices in G corresponding to the lines in $\mathcal{L}^1$. We claim that C forms a multicoloured k-clique in G. Since b_i can only be covered by lines corresponding to the vertices in V_i and $\mathcal{L}^1$ covers $\{b_1, \ldots, b_k\}$, we know that $V(C) \cap V_i \neq \emptyset$. It remains to show that for every pair of vertices in $V(C)$ there exists an edge between them in G. Let v_i denote the vertex in $V(C) \cap V_i$.

Consider all the lines in $\mathcal{L}^1$. Each of these lines are near-horizontal and have exactly $d+1$ red points. Furthermore, no two of them intersect at a red point. Since the total number of red points covered by $\mathcal{L}^1 \cup \mathcal{L}^2$ is at most $2(d+1)k - k^2$, we have that the k lines in $\mathcal{L}^2$ can only cover at most $k(d+1) - k^2$ red points that are not covered by the lines in $\mathcal{L}^1$. That is, the k lines in $\mathcal{L}^2$ contribute at most $k(d+1) - k^2$ *new red points* to the solution. Thus, the number of red points that are covered by both $\mathcal{L}^1$ and $\mathcal{L}^2$ is k^2. Therefore, any two lines l_1 and l_2 such that $l_1 \in \mathcal{L}^1$ and $l_2 \in \mathcal{L}^2$ must intersect at a red point. This implies that either l_1 and l_2 correspond to the same vertex in V or there exists an edge between

the vertices corresponding to them. Let $V(C') = \{w_1, \ldots, w_k\}$ be the set of vertices in G corresponding to the lines in $\mathcal{L}^2$. Since b'_i can only be covered by lines corresponding to the vertices in V_i and $\mathcal{L}^2$ covers $\{b'_1, \ldots, b'_k\}$, we know that $V(C') \cap V_i \neq \emptyset$. Let w_i denote the vertex in V_i such that $l^2_{w_i} \in \mathcal{L}^2$ covers b'_i. We know that $l^1_{v_i}$ and $l^2_{w_i}$ must intersect on a red point. However, by construction no two distinct vertices v_i and w_i belonging to the same vertex class V_i intersect at red point. Thus $v_i = w_i$. This means $C = C'$. This, together with the fact that two lines l_1 and l_2 such that $l_1 \in \mathcal{L}^1$ and $l_2 \in \mathcal{L}^2$ (now lines corresponding to C) must intersect at a red point, implies that C is a multicoloured k-clique in G. □

Since $b = k_\ell = 2k$, Gen-RBSC-lines is W[1]-hard parameterized by k_ℓ or b or $k_\ell + b$. This concludes the proof. □

A closer look at the reduction shows that every set contains exactly one blue point. A natural question to ask is whether the complexity would change if we take the complement of this scenario, that is, each set contains either no blue points or at least two blue points. Shortly, we will see that this implies that the problem becomes FPT. Also, notice that each set in the reduction contains unbounded number of red elements. What about the parameterized complexity if every set in the input contained at most a bounded number, say d, of red elements. Even then the complexity would change but for this we need an algorithm for Gen-RBSC-lines parameterized by $k_\ell + k_r$ that will be presented in Section 6.6.

6.5.2 Special case under the parameter k_ℓ

In this section, we look at the special case when every line in the Gen-RBSC-lines instance contains at least 2 blue points or no blue points at all. We show that in this restricted case, Gen-RBSC-lines is FPT.

Theorem 6.6. Gen-RBSC-lines *parameterized by k_ℓ, where input instances have each set containing either at least* 2 *blue points or no blue points, has a polynomial kernel. There is also an* FPT *algorithm running in $\mathcal{O}(k_\ell^{4k_\ell} \cdot (|U| + |\mathcal{F}|)^{\mathcal{O}(1)})$ time.*

Proof. We exhaustively apply Reduction Rules 6.1, 6.2 and 6.3 to our input instance. In the end, we obtain an equivalent instance that has at least 1 blue point per line. The equivalent instance also has each line containing at least 2 blue points or no blue points. The instance has at most $b = k_\ell^2$ blue points, or else we can correctly say NO. By Observation 6.1 and the assumption on the instance, we can bound ℓ by $\binom{b}{2} \leq k_\ell^4$. Now from Theorem 6.4 we get a polynomial kernel for this special case of Gen-RBSC-lines parameterized by k_ℓ.

Regarding the FPT algorithm, we are allowed to choose at most k_ℓ solution lines from a total of $\ell \leq k_\ell^4$ lines in the instance (of course after we have applied Reduction Rules 6.1, 6.2 and 6.3 exhaustively). For every possible k_ℓ-sized set of lines we check whether the set covers all blue vertices and at most k_r red vertices. If the instance is a YES instance, one such k_ℓ-sized set is a solution family. This algorithm runs in $\mathcal{O}(\binom{k_\ell^4}{k_\ell} \cdot (|U| + |\mathcal{F}|)^{\mathcal{O}(1)}) = \mathcal{O}(k_\ell^{4k_\ell} \cdot (|U| + |\mathcal{F}|)^{\mathcal{O}(1)})$ time. □

6.6 Parameterizing by $k_r + k_\ell$ and $b + k_r$

In the previous sections, we saw that Gen-RBSC-lines parameterized by r is para-NP-complete and is W[1]-hard parameterized by k_ℓ. So there is no hope of an FPT algorithm unless P = NP or FPT =W[1], when parameterized by r and k_ℓ respectively. As a consequence, we consider combining different natural parameters with r to see if this helps to find FPT algorithms. In fact, in this section, we describe a FPT algorithm for Gen-RBSC-lines parameterized by $k_\ell + k_r$. Since $k_r \leq r$, this implies that Gen-RBSC-lines parameterized by $k_\ell + r$ is FPT. This is one of our main technical/algorithmic contribution. Also, since $k_\ell \leq b$ for any minimal solution family of an instance, it follows that Gen-RBSC-lines parameterized by $b + k_r$ belongs to FPT. It is natural to ask whether the Gen-RBSC problem, that is, where sets in the family are arbitrary subsets of the universe and need not correspond to lines, is FPT parameterized by $k_\ell + k_r$. In fact, Theorem 6.15 states that the problem is W[1]-hard even when each set is of size three and contains at least two red points. This shows that indeed restricting ourselves to sets corresponding to lines makes the problem tractable.

We start by considering a simpler case, where the input instance is such that every line contains exactly 1 blue point. Later we will show how we can reduce our main problem to such instances. By the restrictions assumed on the input, no two blue points can be covered by the same line and any solution family must contain at least b lines. Thus, $b \leq k_\ell$ or else, it is a NO instance. Also, a minimal solution family will contain at most $b \leq k_\ell$ lines. Hence, from now on we are only interested in the existence of minimal solution families. In fact, inferring from the above observations, a minimal solution family, in this special case, contains exactly b lines. Let $G_{\mathcal{F}'}$ be the intersection graph that corresponds to a minimal solution $\mathcal{F}'$. Recall, that in $G_{\mathcal{F}'}$ vertices correspond to lines in $\mathcal{F}'$ and there is an edge between two vertices in $G_{\mathcal{F}'}$ if the corresponding lines intersect either at a blue point or a red point. Next, we define notions of *good tuple* and *conformity* which will be useful in designing the FPT algorithm for the special case. Essentially, a good tuple provides a numerical representation of connected components of $G_{\mathcal{F}'}$.

Definition 6.2. *Given an instance $(R, B, \mathcal{F})$ of* Gen-RBSC-lines *we call a tuple*

$$\left(b, p, s, P, \{I'_1, \ldots, I'_s\}, (k_r^1, k_r^2, \ldots, k_r^s)\right)$$

good *if the following hold.*

1. *Integers $p \leq k_r$ and $s \leq b \leq k_\ell$; Here b is the number of blue vertices in the instance.*
2. *$P = P_1 \cup \cdots \cup P_s$ is an s-partition of B;*
3. *For each $1 \leq i \leq s$, I'_i is an ordering for the blue points in part P_i;*
4. *Integers $k_r^i, 1 \leq i \leq s$, are such that $\Sigma_{1 \leq i \leq s} k_r^i = p$.*

Below, we define the relevance of good tuples in the context of our problem.

Definition 6.3. *We say that the minimal solution family $\mathcal{F}'$* conforms *with a good tuple $\left(b, p, s, P, \{I'_1, \ldots, I'_s\}, (k_r^1, k_r^2, \ldots, k_r^s)\right)$ if the following properties hold:*

1. *The components $C_1, \ldots, C_s$ of $G_{\mathcal{F}'}$ give the partition $P = P_1, \ldots, P_s$ on the blue points.*

2. *For each component C_i, $1 \leq i \leq s$, let $t_i = |P_i|$. Let $I'_i = b^i_1, \ldots, b^i_{t_i}$ be an ordering of blue points in P_i. Furthermore assume that $L^i_j \in \mathcal{F}'$ covers the blue point b^i_j. I'_i has the property that, for all $j \leq t_i$, $G_{\mathcal{F}'}[\{L^i_1, \ldots, L^i_j\}]$ is connected. In other words, for all $j \leq t_i$, L^i_j intersects with at least one of the lines from the set $\{L^i_1, \ldots, L^i_{j-1}\}$. Notice that, by minimality of $\mathcal{F}'$, the point of intersection for such a pair of lines is a red point.*

3. *$\mathcal{F}'$ covers $p \leq k_r$ red points.*

4. *In each component C_i, k^i_r is the number of red points covered by the lines in that component. It follows that $\Sigma_{1 \leq i \leq s} k^i_r = p$. In other words, the integers k^i_r form a combination of p.*

The next lemma says that the existence of a minimal solution subfamily $\mathcal{F}'$ results in a conforming good tuple.

Lemma 6.4. *Let $(U, \mathcal{F})$ be an input to* Gen-RBSC-lines *parameterized by $k_\ell + k_r$, such that every line contains exactly 1 blue point. If there exists a solution subfamily $\mathcal{F}'$ then there is a conforming good tuple.*

Proof. Let $\mathcal{F}'$ be a minimal solution family of size $b \leq k_\ell$ that covers $p \leq k_r$ red points. Let $G_{\mathcal{F}'}$ have s components viz. $C_1, C_2, \cdots, C_s$, where $s \leq k_\ell$. For each $i \leq s$, let $\mathcal{F}_{C_i}$ denote the set of lines corresponding to the vertices of $V(C_i)$. $P_i = B \cap \mathcal{F}_{C_i}$, $t_i = |P_i|$ and $k^i_r = |R \cap \mathcal{F}_{C_i}|$. In this special case and by minimality of $\mathcal{F}'$, $|\mathcal{F}_{C_i}| = t_i$. As C_i is connected, there is a sequence $\{L^i_1, L^i_2, \ldots L^i_{t_i}\}$ for the lines in $\mathcal{F}_{C_i}$ such that for all $j \leq t_i$ we have that $G_{\mathcal{F}'}[\{L^i_1, \ldots, L^i_j\}]$ is connected. This means that, for all $j \leq t_i$, L^i_j intersects with at least one of the lines from the set $\{L^i_1, \ldots, L^i_{j-1}\}$. By minimality of $\mathcal{F}'$, the point of intersection for such a pair of lines is a red point. For all $j \leq t_i$, let L^i_j cover the blue point b^i_j. Let $I'_i = b^i_1, b^i_2, \ldots, b^i_{t_i}$. The tuple $\Big(b, p, s, P = P_1 \cup P_2 \ldots \cup P_s, \{I'_1, \ldots, I'_s\}, (k^1_r, k^2_r, \ldots, k^s_r)\Big)$ is a good tuple and it also conforms with $\mathcal{F}'$. This completes the proof. □

The idea of the algorithm is to generate all good tuples and then check whether there is a solution subfamily $\mathcal{F}'$ that conforms to it. The next lemma states we can check for a conforming minimal solution family when we are given a good tuple.

Lemma 6.5. *For a good tuple $(b, p, s, P, \{I'_1, \ldots, I'_s\}, (k^1_r, k^2_r, \ldots, k^s_r))$, we can verify in $\mathcal{O}(b\ell p^b)$ time whether there is a minimal solution family $\mathcal{F}'$ that conforms with this tuple.*

Proof. The algorithm essentially builds a search tree for each partition $P_i, 1 \leq i \leq s$. For each part P_i, we define a set of points R'_i which is initially an empty set.

For each $1 \leq i \leq s$, let $t_i = |P_i|$ and let $I'_i = b^i_1, \ldots, b^i_{t_i}$ be the ordering of blue points in P_i. Our objective is to check whether there is a subfamily $\mathcal{F}'_i \subseteq \mathcal{F}$ such that it covers $b^i_1, \ldots, b^i_{t_i}$, and at most k^i_r red points. At any stage of the algorithm, we have a subfamily $\mathcal{F}'_i$ covering $b^i_1, \ldots, b^i_j$ and at most k^i_r red points. In the next step we try to enlarge $\mathcal{F}'_i$ in such a way that it also covers b^i_{j+1}, but still covers at most k^i_r red points. In some sense we follow the ordering given by I'_i to build $\mathcal{F}'_i$.

Initially, $\mathcal{F}'_i = \emptyset$. At any point of the recursive algorithm we represent the problem to be solved by the following tuple: $(\mathcal{F}'_i,\ R'_i,\ (b^i_j, \ldots, b^i_{t_i}),\ k^i_r - |R'_i|)$. We start the process by

guessing the line in $\mathcal{F}$ that covers b_1^i, say L_1^i. That is, for every $L \in \mathcal{F}$ such that b_1^i is contained in L we recursively check whether there is a solution to the tuple ($\mathcal{F}_i' := \mathcal{F}_i' \cup \{L\}$, $R_i' := R_i' \cup (R \cap L)$, $(b_2^i, \ldots, b_{t_i}^i)$,$k_r^i := k_r^i - |R_i'|$). If any tuple returns YES then we return that there is a subset $\mathcal{F}_i' \subseteq \mathcal{F}$ which covers $b_1^i, \ldots, b_{t_i}^i$, and at most k_r^i red points.

Now suppose we are at an intermediate stage of the algorithm and the tuple we have is ($\mathcal{F}_i'$, R_i', $(b_j^i, \ldots, b_{t_i}^i)$, k_r^i). Let $\mathcal{L}$ be the set of lines such that it contains b_j^i and a red point from R_i'. Clearly, $|\mathcal{L}| \leq |R_i'| \leq k_r^i$. For every line $L \in \mathcal{L}$, we recursively check whether there is a solution to the tuple ($\mathcal{F}_i' := \mathcal{F}_i' \cup \{L\}$, $R_i' := R_i' \cup (R \cap L)$, $(b_{j+1}^i, \ldots, b_{t_i}^i)$,$k_r^i := k_r^i - |R_i'|$). If any tuple returns YES then we return that there is a subset $\mathcal{F}_i' \subseteq \mathcal{F}$ which covers $b_1^i, \ldots, b_{t_i}^i$, and at most k_r^i red points.

Let $\mu = t_i$. At each stage μ drops by one and, except for the first step, the algorithm recursively solves at most k_r^i subproblems. This implies that the algorithm takes at most $\mathcal{O}(|\mathcal{F}|k_r^{t_i}) = \mathcal{O}(\ell k_r^{t_i})$ time.

Notice that the lines in the input instance are partitioned according to the blue points contained in it. Hence, the search corresponding to each part P_i is independent of those in other parts. In effect, we are searching for the components for $G_{\mathcal{F}'}$ in the input instance, in parallel. If for each P_i we are successful in finding a minimal set of lines covering exactly the blue points of P_i while covering at most k_r^i red points, we conclude that a solution family $\mathcal{F}'$ that conforms to the given tuple exists and hence the input instance is a YES instance.

The time taken for the described procedure in each part is at most $\mathcal{O}(\ell k_r^{t_i})$. Hence, the total time taken to check if there is a conforming minimal solution family $\mathcal{F}'$ is at most

$$\mathcal{O}(\ell \cdot \sum_{i=1}^{s} k_r^{t_i}) = \mathcal{O}(s\ell p^b) = \mathcal{O}(b\ell p^b).$$

This concludes the proof. □

We are ready to describe our FPT algorithm for this special case of Gen-RBSC-lines parameterized by $k_\ell + k_r$.

Lemma 6.6. *Let $(U, \mathcal{F}, k_\ell, k_r)$ be an input to* Gen-RBSC-lines *such that every line contains exactly* 1 *blue point. Then we can check whether there is a solution subfamily $\mathcal{F}'$ to this instance in time $k_\ell^{\mathcal{O}(k_\ell)} \cdot k_r^{\mathcal{O}(k_r)} \cdot (|U| + |\mathcal{F}|)^{\mathcal{O}(1)}$ time.*

Proof. Lemma 6.4 implies that for the algorithm all we need to do is to enumerate all possible good tuples $(b, p, s, P, \{I_1', \ldots, I_s'\}, (k_r^1, k_r^2, \ldots, k_r^s))$, and for each tuple, check whether there is a conforming minimal solution family. Later, we use the algorithm described in Lemma 6.5. We first give an upper bound on the number of tuples and how to enumerate them.

1. There are k_ℓ choices for s and k_r choices for p.
2. There can be at most b^{k_ℓ} choices for P which can be enumerated in $\mathcal{O}(b^{k_\ell} \cdot k_\ell)$ time.
3. For each $j \leq s$, I_j' is ordering for blue points in P_i. Thus, if $|P_i| = t_i$, then the number of ordering tuples $\{I_1', \ldots, I_s'\}$ is upper bounded by $\prod_{i=1}^{s} t_i! \leq \prod_{i=1}^{s} t_i^{t_i} \leq \prod_{i=1}^{s} b^{t_i} = b^b$. Such orderings can be enumerated in $\mathcal{O}(b^b)$ time.

4. For a fixed $p \leq k_r, s \leq k_\ell$, there are at most $\binom{p+s-1}{s-1}$ solutions for $k_r^1+k_r^2+\ldots+k_r^s = p$ and this set of solutions can be enumerated in $\mathcal{O}(\binom{p+s-1}{s-1} \cdot ps)$ time. Notice that if $p \geq s$ then the time required for enumeration is $\mathcal{O}((2p)^p \cdot ps)$. Otherwise, the required time is $\mathcal{O}((2s)^s \cdot ps)$. As $p \leq k_r$ and $s \leq k_\ell$, the time required to enumerate the set of solutions is $\mathcal{O}(k_\ell^{\mathcal{O}(k_\ell)} k_r^{\mathcal{O}(k_r)} \cdot k_\ell k_r)$.

Thus we can generate the set of tuples in time $k_\ell^{\mathcal{O}(k_\ell)} \cdot k_r^{\mathcal{O}(k_r)}$.Using Lemma 6.5, for each tuple we check in at most $\mathcal{O}(k_r^{k_\ell} \cdot k_\ell \ell)$ time whether there is a conforming solution family or not. If there is no tuple with a conforming solution family, we know that the input instance is a NO instance. The total time for this algorithm is $k_\ell^{\mathcal{O}(k_\ell)} k_r^{\mathcal{O}(k_r)} k_r^{\mathcal{O}(k_\ell)} \cdot (|U| + |\mathcal{F}|)^{\mathcal{O}(1)}$. Again, if $k_r \leq k_l$ then $k_r^{\mathcal{O}(k_\ell)} = k_\ell^{\mathcal{O}(k_\ell)}$. Otherwise, $k_r^{\mathcal{O}(k_\ell)} = k_r^{\mathcal{O}(k_r)}$. Either way, it is always true that $k_r^{\mathcal{O}(k_\ell)} = k_\ell^{\mathcal{O}(k_\ell)} k_r^{\mathcal{O}(k_r)}$. Thus, we can simply state the running time to be $k_\ell^{\mathcal{O}(k_\ell)} \cdot k_r^{\mathcal{O}(k_r)} \cdot (|U| + |\mathcal{F}|)^{\mathcal{O}(1)}$. □

We return to the general problem of Gen-RBSC-lines parameterized by $k_\ell+k_r$. Instances in this problem may have lines containing 2 or more blue points. We use the results and observations described above to arrive at an FPT algorithm for Gen-RBSC-lines parameterized by $k_\ell + k_r$.

Theorem 6.7. Gen-RBSC-lines *parameterized by* $k_\ell + k_r$ *is* FPT*, with an algorithm that runs in* $k_\ell^{\mathcal{O}(k_\ell)} \cdot k_r^{\mathcal{O}(k_r)} \cdot (|U| + |\mathcal{F}|)^{\mathcal{O}(1)}$ *time.*

Proof. Given an input $(U, \mathcal{F}, k_\ell, k_r)$ for Gen-RBSC-lines parameterized by $k_\ell + k_r$, we do some preprocessing to make the instance simpler. We exhaustively apply Reduction Rules 6.1, 6.2 and 6.3. After this, by Observation 6.2, the reduced equivalent instance has at most $\binom{k_\ell}{2}$ blue points if it is a YES instance.

A minimal solution family can be broken down into two parts: the set of lines containing at least 2 blue points, and the remaining set of lines which contain exactly 1 blue point. Let us call these sets $\mathcal{F}_2$ and $\mathcal{F}_1$ respectively. We start with the following observation.

Observation 6.3. *Let* $\mathcal{F}'' \subseteq \mathcal{F}$ *be the set of lines that contain at least* 2 *blue points. There are at most* $\binom{k_\ell^4}{k_\ell}$ *ways in which a solution family can intersect with* $\mathcal{F}''$*.*

Proof. Since $b \leq \binom{k_\ell}{2}$, it follows from Observation 6.1 that $|\mathcal{F}''| \leq k_\ell^4$. For any solution family, there can be at most k_ℓ lines containing at least 2 blue points. Since the number of subsets of $\mathcal{F}''$ of size at most k_ℓ is bounded by $k_\ell^{4k_\ell}$, the observation is true. □

From Observation 6.3, there are $k_\ell^{4k_\ell}$ choices for the set of lines in $\mathcal{F}_2$. We branch on all these choices of $\mathcal{F}_2$. On each branch, we reduce the budget of k_ℓ by the number of lines in $\mathcal{F}_2$ and the budget of k_r by $|R \cap \mathcal{F}_2|$. Also, we make some modifications on the input instance: we delete all other lines containing at least 2 blue points from the input instance. We delete all points of U covered by $\mathcal{F}_2$ and all lines passing through blue points covered by $\mathcal{F}_2$. Our modified input instance in this branch now satisfies the assumption of Lemma 6.6 and we can find out in $k_\ell^{\mathcal{O}(k_\ell)} k_r^{\mathcal{O}(k_r)} \cdot (|U| + |\mathcal{F}|)^{\mathcal{O}(1)}$ time whether there is a minimal solution family $\mathcal{F}_1$ for this reduced instance. If there is, then $\mathcal{F}_2 \cup \mathcal{F}_1$ is a minimal solution for our original input instance and we correctly say YES. Thus the total running time of this algorithm is $k_\ell^{\mathcal{O}(k_\ell)} \cdot k_r^{\mathcal{O}(k_r)} \cdot (|U| + |\mathcal{F}|)^{\mathcal{O}(1)}$.

It may be noted here that for a special case where we can use any line in the plane as part of the solution, the second part of the algorithm becomes considerably simpler. Here for each blue point b, we can use an arbitrary line containing only b and no red point. □

Corollary 6.1. Gen-RBSC-lines *parameterized by $k_\ell + d$, where every line contains at most d red points, is* FPT. *The running time of the* FPT *algorithm is $(dk_\ell)^{\mathcal{O}(dk_\ell)} \cdot (|U| + |\mathcal{F}|)^{\mathcal{O}(1)}$. The problem remains* FPT *for all parameter sets Γ' that contain $\{k_\ell, d\}$ or $\{b, d\}$.*

Proof. In this special case, any solution family can contain at most dk_ℓ red points. Hence we can safely assume that $k_r \leq dk_\ell$ and apply Theorem 6.7. □

6.6.1 Kernelization for Gen-RBSC-lines parameterized by $k_\ell + k_r$ and $b + k_r$

We give a polynomial parameter transformation from Set Cover parameterized by universe size n, to Gen-RBSC-lines parameterized by $k_\ell + k_r + b$. Proposition 6.1(ii) implies that on parameterizing by any subset of the parameters $\{k_\ell, k_r, b\}$, we will also obtain a negative result for polynomial kernels.

Theorem 6.8. Gen-RBSC-lines *parameterized by $k_\ell + k_r + b$ does not allow a polynomial kernel unless* CoNP $\subseteq$ NP/*poly.*

Proof. Let $(U, \mathcal{S})$ be a given instance of Set Cover. Let $|U| = n, |\mathcal{S}| = m$. We construct an instance $(R \cup B, \mathcal{F})$ of Gen-RBSC-lines as follows. We assign a blue point $b_u \in B$ for each element $u \in U$ and a red point $r_S \in R$ for each set $S \in \mathcal{S}$. The red and blue points are placed such that no three points are collinear. We add a line between b_u and r_S if $u \in S$ in the Set Cover instance. Thus the Gen-RBSC-lines instance $(R \cup B, \mathcal{F})$ that we have constructed has $b = n$, $r = m$ and $\ell = \sum_{S \in \mathcal{S}} |S|$. We set $k_r = k$ and $k_\ell = n$.

Claim 6.2. *All the elements in $(U, \mathcal{S})$ can be covered by k sets if and only if there exist n lines in $(R \cup B, \mathcal{F})$ that contain all blue points but only k red points.*

Proof. Suppose $(U, \mathcal{S})$ has a solution of size k, say $\{S_1, S_2, \cdots S_k\}$. The red points in the solution family for Gen-RBSC-lines are $\{r_{S_1}, r_{S_2}, \cdots r_{S_k}\}$ corresponding to $\{S_1, S_2, \cdots S_k\}$. For each element $u \in U$, we arbitrarily assign a covering set S_u from $\{S_1, S_2, \cdots S_k\}$. The solution family is the set of lines defined by the pairs $\{(b_u, r_{S_u}) \mid u \in U\}$. This covers all blue points.

Conversely, if $(R \cup B, \mathcal{F})$ has a solution family $\mathcal{F}'$ covering k red points and using at most n lines, the sets in $\mathcal{S}$ corresponding to the red points in $\mathcal{F}'$ cover all the elements in $(U, \mathcal{S})$. □

If $k > n$, then the Set Cover instance is a trivial YES instance. Hence, we can always assume that $k \leq n$. This completes the proof that Gen-RBSC-lines parameterized by $k_\ell + k_r + b$ cannot have a polynomial sized kernel unless CoNP $\subseteq$ NP/poly. □

6.7 Hyperplanes: parameterized by $k_\ell + k_r$

Theorem 6.9. GEN-RBSC *for hyperplanes in* $\mathbb{R}^d$*, for a fixed positive integer* d*, is* W[1]*-hard when parameterized by* $k_\ell + k_r$*.*

Proof. The proof of hardness follows from a reduction from k-CLIQUE problem. The proof follows a framework given in [Marx 2006].

Let (G, k) be an instance of k-CLIQUE problem. Our construction consists of a $k \times k$ matrix of gadgets G_{ij}, $1 \leq i, j, \leq k$. Consecutive gadgets in a row are connected by horizontal connectors and consecutive gadgets in a column are connected by vertical connectors. Let us denote the horizontal connector connecting the gadgets G_{ij} and G_{ih} as $H_{i(jh)}$ and the vertical connector connecting the gadgets G_{ij} and G_{hj} as $V_{(ih)j}$, $1 \leq i, j, h \leq k$.
Gadgets: The gadget G_{ij} contains a blue point b_{ij} and a set R_{ij} of $d-2$ red points. In addition there are n^2 sets $R'_{ij}(a, b), 1 \leq a, b \leq n$, each having two red points each.
Connectors: The horizontal connector $H_{i(jh)}$ has a blue point $b_{i(jh)}$ and a set $R_{i(jh)}$ of $d-2$ red points. Similarly, the vertical connector $V_{(ih)j}$ a blue point $b_{(ih)j)}$ and a set $R_{(ih)j}$ of $d-2$ red points.

The points are arranged in general position i.e., no set of $d+2$ points lie on the same d-dimensional hyperplane. In other words, any set of $d+1$ points define a distinct hyperplane.
Hyperplanes: Assume $1 \leq i, j, h \leq k$ and $1 \leq a, b, c \leq n$. Let $P_{ij}(a, b)$ be the hyperplane defined by the $d+1$ points of $b_{ij} \cup R_{ij} \cup R'_{ij}(a, b)$. Let $P^h_{i(jh)}(a, b, c)$ be the hyperplane defined by $d+1$ points of $b_{i(jh)} \cup R_{i(jh)} \cup r_1 \cup r_2$ where $r_1 \in R'_{ij}(a, b)$ and $r_2 \in R'_{ih}(a, c)$. Let $P^v_{(ij)h}(a, b, c)$ be the hyperplane defined by $d+1$ points of $b_{(ij)h)} \cup R_{(ij)h} \cup r_1 \cup r_2$ where $r_1 \in R'_{ih}(a, c)$ and $r_2 \in R'_{jh}(b, c)$.
For each edge $ab \in E(G)$, we add $k(k-1)$ hyperplanes of the type $P_{ij}(a, b)$, $i \neq j$. Further, for all $1 \leq a \leq n$, we add k hyperplanes of the type $P_{ii}(a, a)$, $1 \leq i \leq k$. The hyperplane $P^h_{i(jh)}(a, b, c)$ containing the blue point $b_{i(jh)}$ in a horizontal connector, is added to the construction if $P_{ij}(a, b)$ and $P_{ih}(a, c)$ are present in the construction. Similarly, the hyperplane $P^v_{(ij)h}(a, b, c)$ containing the blue point $b_{(ij)h}$ in a vertical connector, is added to the construction if $P_{ih}(a, c)$ and $P_{jh}(b, c)$ are present in the construction.

Thus our construction has $k^2 + 2k(k-1)$ blue points, $(k^2 + 2k(k-1))(d-2) + 2n^2k^2$ red points and $O((m^2k^2))$ hyperplanes.

Claim 6.3. *G has a k-clique if and only if all the blue points in the constructed instance can be covered by $k^2 + 2k(k-1)$ hyperplanes covering at most $k^2d + 2k(k-1)(d-2)$ red points.*

Proof. Assume G has a clique of size k and let $\{a_1, a_2, \cdots, a_k\}$ be the vertices of the clique. Now we show a set cover of desired size exists. Choose k hyperplanes, $P_{ii}(a_i, a_i), 1 \leq i \leq k$, to cover the diagonal gadgets. To cover other gadgets,G_{ij}, choose the hyperplanes $P_{ij}(a_ia_j)$ and to cover the connectors, $H_{i(jh)}$ and $V_{(ih)j}$, choose the hyperplanes $P^h_{i(jh)}(a_i, a_j, a_h)$ and $P^v_{(ij)h}(a_i, a_j, a_h)$. The fact that $\{a_1, a_2, \cdots, a_k\}$ forms a clique implies that these hyperplanes do exist in the construction.

Now assume a set cover of given size exists. To cover the blue point b_{ij} in the gadget G_{ij}, any hyperplane adds d red points. Also to cover the blue point in each connector, we need to add $d-2$ extra red points. Since each hyperplane contains d red points and we

have already used up our budget of red points, each hyperplane covering the connector points should reuse two red points that have been used in covering gadgets. By construction, this is possible only when all gadgets in a row(column) are covered by hyperplanes corresponding to edges incident on the same vertex viz. the vertex corresponding to the hyperplane covering the diagonal gadget in the row(column). This implies that G has a required clique. □

This completes the proof. □

6.8 Multivariate complexity of Gen-RBSC-lines: Proof of Theorem 6.1

The first part of Theorem 6.1 (parameterized complexity dichotomy) follows from Theorems 6.3, 6.4, 6.5 and 6.7. Recall that $\Gamma = \{\ell, r, b, k_\ell, k_r\}$. To show the kernelization dichotomy of the parameterizations of Gen-RBSC-lines that admit FPT kernels we do as follows:

- Show that the problem admits a polynomial kernel parameterized by ℓ (Theorem 6.4). This implies that for all Γ' that contains ℓ, the parameterization admits a polynomial kernel.
- Show that the problem does not admit a polynomial kernel when parameterized by $k_\ell + k_r + b$ (Theorem 6.8). This implies that for all subsets of $\{k_\ell, k_r, b\}$, the parameterization does not allow a polynomial kernel.
- The remaining FPT variants of Gen-RBSC-lines correspond to parameter sets Γ' that contain either r or $\{r, b\}$ together. Recall that, $k_r \leq r$ and $k_\ell \leq b$. The two smallest combined parameters for which we can not infer the kernelization complexity from Theorem 6.8 are $r + k_\ell$ and $r + b$. We show below (Theorem 6.10) that Gen-RBSC admits a quadratic kernel parameterized by $r + k_\ell$. Since in any minimal solution family $k_\ell \leq b$, this also implies a quadratic kernel for the parameterization $r + b$. Thus, if parameterization by a set Γ', which contains either r or $\{r, b\}$, allows an FPT algorithm then it also allows a polynomial kernel.

Theorem 6.10. Gen-RBSC-lines *parameterized by $k_\ell + r$ admits a polynomial kernel.*

Proof. Given an instance of Gen-RBSC-lines we first exhaustively apply Reduction Rules 6.1, 6.2 and 6.3 and obtain an equivalent instance. By Observation 6.2, the reduced instance has at most $b \leq k_\ell^2$ blue points. By Observation 6.1, the number of lines containing at least two points is $\binom{r+b}{2}$. After applying Reduction Rule 6.1, there are no lines with only one red point. Also, for a blue point b_i, if there are many lines that contain only b_i, then we can delete all but one of those lines. Therefore, the number of lines that contain exactly one point is bounded by b. Thus, we get a kernel of k_ℓ^2 blue points, $\binom{r+k_\ell^2}{2} + k_\ell^2$ lines and r red points. This concludes the proof. □

Combining Theorems 6.4, 6.8 and 6.10 and the discussion above we prove the second part of the Theorem 6.1 (kernelization dichotomy).

6.9 Parameterized Landscape for Red Blue Set Cover with lines

Until now our main focus was the Gen-RBSC-lines problem. In this section, we study the original RBSC-lines problem. Recall that the original RBSC-lines problem differs from the Gen-RBSC-lines problem in the following way – here our objective is *only* to minimize the number of red points that are contained in a solution subfamily, and *not* the size of the subfamily itself. That is, $k_\ell = |\mathcal{F}|$. This change results in a slightly different landscape for RBSC-lines compared to Gen-RBSC-lines. As before let $\Gamma = \{\ell, r, b, k_\ell, k_r\}$. We first observe that for all those $\Gamma' \subseteq \Gamma$ that do not contain k_ℓ as a parameter and Gen-RBSC-lines is FPT parameterized by Γ', RBSC-lines is also FPT parameterized by Γ'. Next we list out the subsets of parameters for which the results do not follow from the result on Gen-RBSC-lines.

- RBSC-lines becomes FPT parameterized by r.
- W[2]-hard parameterized by k_r.

6.9.1 RBSC-lines parameterized by r

Theorem 6.11. RBSC-lines *parameterized by r is* FPT. *Furthermore,* RBSC-lines *parameterized by r does not allow a polynomial kernel unless* CoNP $\subseteq$ NP/*poly.*

Proof. We proceed by enumerating all possible k_r-sized subsets of R. For each subset, we can check in polynomial time whether the lines spanned by exactly those points cover all blue points. This is our FPT algorithm, which runs in $\mathcal{O}(2^r \cdot (|U| + |\mathcal{F}|)^{\mathcal{O}(1)})$.

Using Proposition 6.1, it is enough to show a polynomial parameter transformation from Set Cover parameterized by size m of the set family, to RBSC-lines parameterized by r. The reduction is exactly the same as the one given in the proof of Theorem 6.8. This gives the desired second part of the theorem. □

6.9.2 RBSC-lines parameterized by k_r

Here, we study parameterization by k_r and some special cases which lead to FPT algorithm. We prove that RBSC-lines parameterized by k_r is W[2]-hard. From Proposition 6.1, Set Cover parameterized by solution family size k is W[2]-hard. The W[2]-hardness of RBSC-lines parameterized by k_r can be proved by a many-one reduction from Set Cover parameterized by k. The reduction is exactly the one that is given in Theorem 6.8.

Theorem 6.12. RBSC-lines *parameterized by k_r is* W[2]*-hard.*

FPT result under special assumptions

In this section, we consider a special case, where in the given instance every line contains either no red points or at least 2 red points. There are two reasons motivating the study

of this special case. Firstly, in the W[2]-hardness proof we crucially used the fact that the constructed RBSC-LINES instance has a set of lines with exactly 1 red point. Thus, it is necessary to check if this is the reason leading to the hardness of the problem. Secondly, if we look at RBSC (sets in the family can be arbitrary) parameterized by k_r and assumed that in the given instance every line contains either no red points or at least 2 red points, then too the problem is W[1]-hard (see Theorem 6.15). However, when we consider RBSC-LINES parameterized by k_r and where in the given instance every set contains either no red points or at least 2 red points, the problem is FPT.

For our algorithm we also need the following new reduction rule.

Reduction Rule 6.4. *If there is a set $S \in \mathcal{F}$ with only blue points then delete that set from $\mathcal{F}$ and include the set in the solution.*

Lemma 6.7. *Reduction Rule 6.4 is safe.*

Proof. Since the parameter is k_r, there is no size restriction on the number of lines in the solution subfamily $\mathcal{F}'$. If $\mathcal{F}'$ is a solution subfamily and $S \in \mathcal{F}$ then under this parameterization, $\mathcal{F}' \cup \{S\}$ is also a solution family covering all blue points and at most k_r red points. This shows that Reduction Rule 6.4 is valid. □

Theorem 6.13. RBSC-LINES *parameterized by k_r, where the input instance has every set containing at least 2 red points or no red points at all, has an algorithm with running time $k_r^{\mathcal{O}(k_r^2)} \cdot (|U| + |\mathcal{F}|)^{\mathcal{O}(1)}$.*

Proof. Given an instance of RBSC-LINES, we first exhaustively apply Reduction Rules 6.1, 6.2 and 6.4 and obtain an equivalent instance. At the end of these reductions we obtain an equivalent instance where every line has at least 1 blue point and at least 2 red points, but at most k_r red points.

Suppose $\mathcal{F}'$ is a solution family. Since a line with a red point has at least 2 red points, by Observation 6.1, the total number of sets that can contain the red points covered by $\mathcal{F}'$ is at most $\binom{k_r}{2}$. This means that, if the input instance is a YES instance, there exists a solution family with at most $k_\ell = \binom{k_r}{2}$ lines. Now we can apply the algorithm for GEN-RBSC-LINES parameterized by $k_\ell + k_r$ described in Theorem 6.7 to obtain an algorithm for RBSC-LINES parameterized by k_r. □

Theorem 6.13 gives an FPT algorithm for RBSC-LINES parameterized by k_r. In what follows we show that the same parameterization does not yield a polynomial kernel for this special case of RBSC-LINES. Towards this we give a polynomial parameter transformation from SET COVER parameterized by universe size n, to RBSC-LINES parameterized by k_r and under the assumption that all sets in the input instance have at least 2 red points.

Theorem 6.14. RBSC-LINES *parameterized by k_r, and under the assumption that all lines in the input have at least 2 red points, does not allow a polynomial kernel unless* CoNP $\subseteq$ NP/*poly.*

Proof. Let $(U, \mathcal{S})$ be a given instance of the SET COVER problem. We construct an instance $(R \cup B, \mathcal{F})$ of RBSC-LINES as follows. We assign a blue point $b_u \in B$ for each element $u \in U$ and a red point $r_S \in R$ for each set $S \in \mathcal{S}$. The red and blue points are placed

such that no three points are collinear. We add a line between b_u and r_S if $u \in S$ in the Set Cover instance. To every line L, defined by a blue point b_u and a red points r_S, we add a unique red point $r_L \in R$. Thus the RBSC-lines instance $(R \cup B, \mathcal{F})$ that we have constructed has n blue points, $\sum_{s \in \mathcal{S}} |S|$ lines and $m + \sum_{s \in \mathcal{S}} |S|$ red points. We set $k_r = k + n$.

Claim 6.4. *All the elements in $(U, \mathcal{S})$ can be covered by k sets if and only if there exist lines in $(R \cup B, \mathcal{F})$ that contain all blue points but only $k + n$ red points.*

Proof. Suppose $(U, \mathcal{S})$ has a solution of size k, say $\{S_1, S_2, \cdots S_k\}$. To each element $u \in U$, we arbitrarily associate a covering set S_u from $\{S_1, S_2, \cdots S_k\}$. Our solution family $\mathcal{F}'$ of lines are the lines defined by the pairs of points $\{(b_u, r_{S_u}) \mid u \in U\}$. These lines cover all blue points. The number of red points contained in these lines are the k red points $\{r_{S_1}, r_{S_2}, \cdots r_{S_k}\}$ associated with $\{S_1, S_2, \cdots S_k\}$, and the n red points $\{r_L \mid L \in \mathcal{F}'\}$. Therefore, in total there are $k + n$ red points in the solution.

Conversely, suppose $(R \cup B, \mathcal{F})$ has a family $\mathcal{F}'$ covering all blue points and at most $k + n$ red points. The construction ensures that at least n lines are required to cover the n blue points. This also implies that the unique red points belonging to each of these lines add to the number of red points contained in the solution family. The remaining k red points, that are contained in the solution family, correspond to sets in $\mathcal{S}$ that cover all the elements in $(U, \mathcal{S})$. □

If $k > n$, then the Set Cover instance is a trivial YES instance. Hence, we can always assume that $k \leq n$. This completes the proof that RBSC-lines parameterized by k_r, and under the assumption that every line in the input instance has at least 2 red points, cannot have a polynomial sized kernel unless CoNP $\subseteq$ NP/poly. □

6.9.3 Proof of Theorem 6.2

The proof of Theorem 6.2 follows from Theorems 6.1, 6.11 and 6.12.

6.10 Generalised Red Blue Set Cover

In this section, we show that for several parameterizations, under which Gen-RBSC-lines is FPT, the Gen-RBSC problem is not. In this section we give the following three results which complement the corresponding results in the geometric setting.

1. Gen-RBSC is W[1]-hard parameterized by $k_\ell + k_r$ when every set has size at most three and contains at least two red elements.

2. Gen-RBSC is W[2]-hard parameterized by $k_\ell + r$ when every set contains at most one red element.

3. Gen-RBSC is FPT, parameterized by k_ℓ and d, when every set has at most one red element. Here, d is the size of the maximum cardinality set in $\mathcal{F}$.

6.10.1 Gen-RBSC parameterized by $k_\ell + k_r$ and $k_\ell + r$

Theorem 6.15. Gen-RBSC *is W-hard in the following cases:*

1. *When every set contains at least two red elements but at most three elements, and the parameters are* $\{k_\ell, k_r\}$*, the problem is* W[1]*-hard.*
2. *When every set contains at most one red element and the parameters are* $\{k_\ell, r\}$*, then the problem is* W[2]*-hard.*

Proof. We start by proving the first result. From an instance (G, k) of Multicoloured Clique parameterized by k, we construct an instance $(U = (R, B), \mathcal{F})$ of Gen-RBSC parameterized by $k_\ell + k_r$ with the restriction that the size of each set is at most three and there are at least 2 red elements. The construction is as follows.

- Let the given vertex set be $V(G) = V_1 \uplus V_2 \uplus \ldots \uplus V_k$. For every pair (i, j), $1 \leq i < j \leq k$, we introduce a new blue element $b_{ij} \in B$. Thus we have $\binom{k}{2}$ blue elements.
- For each vertex $v \in V(G)$ we introduce a new red element $r_v \in R$.
- $U = R \uplus B$.
- For each $e = uv \in E(G)$ such that $u \in V_i, v \in V_j$ and $i < j$, we define a set $S_e \in \mathcal{F}$ which contains the elements $\{b_{ij}, r_u, r_v\}$.
- We set $k_r = k$ and $k_\ell = \binom{k}{2}$.

This completes our construction. Notice that every set in $\mathcal{F}$ has at least 2 red elements and has size exactly three.

First, assume that (G, k) is a YES instance. Then there is a k-sized multicoloured clique C in G. Let $E(C)$ denote the set of edges of C. Pick the subfamily $\mathcal{F}' = \{S_e \mid e \in E(C)\}$ of size $\binom{k}{2}$. Since C is a multicoloured clique, for all (i, j), $1 \leq i < j \leq k$ there is an edge $e_{ij} \in E(C)$ whose endpoints belong to V_i and V_j. Consequently, there is a set $S_{e_{ij}} \in \mathcal{F}'$ that contains b_{ij}. The total number of red elements contained in $\mathcal{F}'$ is equal to the size $|V(C)| = k$. This shows that $(U, \mathcal{F}, k)$ is a YES instance of Gen-RBSC.

Conversely, suppose $(U, \mathcal{F})$ is a YES instance of Gen-RBSC. Let $\mathcal{F}'$ be a minimal subfamily of at most $\binom{k}{2}$ sets that covers at most k red elements. Let $V(C)$ be the vertices in G corresponding to the red elements in $\mathcal{F}'$. Notice that there are $\binom{k}{2}$ blue elements, no two of which can be covered by the same set. Thus, for all (i, j), $1 \leq i < j \leq k$, $\mathcal{F}'$ must contain exactly one set $S_e = \{b_{ij}, r_1^{ij}, r_2^{ij}\}$. This implies that for every i, $1 \leq i \leq k$ the sets in $\mathcal{F}'$ must contain a red element corresponding to a vertex in V_i. Hence, for all $i, 1 \leq i \leq k$, $C \cap V_i \neq \emptyset$. Also, C forms a clique since the set $S_e = \{b_{ij}, r_1^{ij}, r_2^{ij}\}$ corresponds to the edge between the vertices selected from V_i and V_j. Therefore, (G, k) is a YES instance of Multicoloured Clique. This proves that Gen-RBSC, parameterized by $k_\ell + k_r$, is W[1]-hard under the said assumption.

For the second part of the statement, observe that Set Cover is a special case of this problem and therefore, the problem is W[2]-hard. □

6.10.2 A special case of Gen-RBSC parameterized by k_ℓ

In this section, we restrict the input instances to those where every set has at most 1 red element and at most d blue elements. We design an FPT algorithm for this special case of Gen-RBSC parameterized by $k_\ell + d$. It is reasonable to assume that there is *no set* in the given instance with only red elements, since Reduction Rule 6.1 can be applied to obtain an equivalent instance of Gen-RBSC, under the parameters of $\{k_\ell, d\}$.

We were able to show that this problem has an FPT algorithm. However, it was pointed out to us by an anonymous reviewer that there is a simple algorithm based on Dynamic Programming technique. Thus, in the following two subsections, we present the two algorithms.

An algorithm based on Subgraph Isomorphism

We obtain the desired algorithm by making $2^{\mathcal{O}(dk_\ell)}$ instances of the Subgraph Isomorphism (Sub-Iso) problem where the pattern graph has size $\mathcal{O}(dk_\ell)$ and treewidth 2, and the given instance is a YES instance if and only if one of the constructed instances is a YES instance of the Sub-Iso problem. To solve Sub-Iso we use known algorithms. We need the following definitions to describe our algorithm.

Definition 6.4. *Two graphs G_1 and G_2 are said to be isomorphic if there is a function $f : V(G_1) \to V(G_2)$ that satisfies the following properties:*

1. *f is a bijective function, i.e., f^{-1} is a function from $V(G_2)$ to $V(G_1)$;*
2. *for all $e = uv \in E(G_1)$, $f(u)f(v) \in E(G_2)$.*

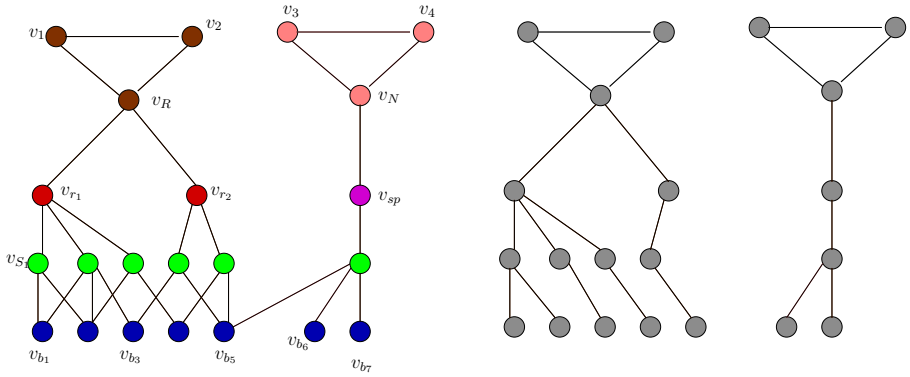
The function f is called an isomorphism function. *This function can be extended to sets of vertices analogously. That is, for all $V_1' \subseteq V(G_1), f(V_1') = \{f(v) \mid v \in V_1'\} \subseteq V(G_2)$. We denote the two isomorphic graphs as $G_1 \simeq G_2$.*

The Subgraph Isomorphism problem is formally defined as follows.

Subgraph Isomorphism (Sub-Iso)	**Parameter:** $\lvert H\rvert$
Input: A host graph G and a pattern graph H	
Question: Is there a subgraph $G' \subseteq G$ such that $H \simeq G'$?	

For our reduction to Sub-Iso given an input instance $(U = (R, B), \mathcal{F})$ we form the host graph G as follows.

- Add an independent set $V_\mathcal{F} = \{v_S \mid S \in \mathcal{F}\}$ to $V(G)$.
- Add independent sets $V_B = \{v_{b'} \mid b' \in B\}$ and $V_R = \{v_{r'} \mid r' \in B\}$ to $V(G)$.
- For a pair of vertices $v_S \in V_\mathcal{F}$ and $v_{b'} \in V_B$, add an edge between them if $b' \in S$.
- For a pair of vertices $v_S \in V_\mathcal{F}$ and $v_{r'} \in V_R$, add an edge between them if $r' \in S$. However, we will say this slightly differently to make some arguments in the



a) Host graph

b) Pattern Graphs

Figure 6.3: Illustration of our reduction to Sub-Iso. The instance shown above is obtained after applying the reduction to the instance $R = \{r_1, r_2\}, B = \{b_1, b_2, b_3, b_4, b_5, b_6, b_7\}$, $\mathcal{F} = \{S_1 = \{r_1, b_1, b_2\}, S_2 = \{r_1, b_1, b_2, b_3\}, S_3 = \{r_1, b_2, b_4\}, S_4 = \{r_2, b_3, b_5\}, S_5 = \{r_2, b_4, b_5\}, S_6 = \{b_5, b_6, b_7\}\}$. The right hand side is one pattern graph when $k_\ell = 5$ and $k_r = 2$. It is also an example of a $(2, 4, 5)$-pattern and a $(1, 1, 2)$ pattern.

upcoming proofs simpler. For the collection of sets that contain the same red element r', we add a common neighbour $v_{r'}$ to the corresponding vertices in $V_{\mathcal{F}}$. Also, for the collection of sets that do not contain any red elements, we add a common neighbour v_{sp} to the corresponding vertices in $V_{\mathcal{F}}$.

- We add a vertex v_R, which is a common neighbour to the vertices of V_R.
- We add a vertex v_N and an edge between v_N and v_{sp}.
- We add new vertices v_1, v_2, v_3, v_4 and add edges such that $\{v_1, v_2, v_R\}$ form a triangle and $\{v_3, v_4, v_N\}$ form a triangle.

This completes the construction of the host graph G. Notice that only the vertices of $\{v_R, v_1, v_2, v_3, v_4, v_N\}$ participate in a triangle. See Figure 6.3 for an illustration of construction of the host graph.

Now we give some definitions that will be useful in describing the pattern graph of constant treewidth.

Definition 6.5. *An (a_1, a_2, a_3)-forest, $a_1 \leq a_2 \leq a_3$, is a forest with the following properties:*

1. *there are a_1 components;*
2. *each component i is rooted at a vertex u_i. The distance between u_i and any connected leaf is exactly 2. Let $\tilde{U} = \{u_i \mid 1 \leq i \leq a_1\}$;*
3. *$\Sigma_{1 \leq i \leq a_1} |N(u_i)| = a_2$;*
4. *$\Sigma_{1 \leq i \leq a_1} |(N(N(u_i)) \setminus \{u_i\})| = a_3$ (that is, the total number of leaves in the (a_1, a_2, a_3) forest is a_3).*

We will always think of an (a_1, a_2, a_3)-forest as a forest where each component (tree) is rooted at some vertex in $\tilde{U}$.

For a forest H, we denote the set $\tilde{U}$ of root vertices as $\tilde{U}_H$. From item 2 of the definition of (a_1, a_2, a_3)-forests, if two (a_1, a_2, a_3)-forests H_1 and H_2 are *isomorphic*, there is an isomorphism function f such that $f(\tilde{U}_{H_1}) = \tilde{U}_{H_2}$. Let $\mathcal{Z}$ be the set of non-isomorphic (a_1, a_2, a_3)-forests. The next lemma gives an upper bound on $|\mathcal{Z}|$ and also shows how this set of forests can be enumerated efficiently.

Lemma 6.8. *For a tuple (a_1, a_2, a_3), there can be at most $2^{2(a_2+a_3)}$ (a_1, a_2, a_3)-forests. That is, $|\mathcal{Z}| \leq 2^{2(a_2+a_3)}$. This set can be enumerated in $\mathcal{O}(2^{2(a_2+a_3)} \cdot (a_1a_2 + a_2a_3))$ time.*

Proof. Informally, we can give an upper bound as follows. For $1 \leq i \leq a_1$, let x_i denote a the number of possible children of u_i in an (a_1, a_2, a_3)-forest. Then clearly, $\sum_{i=1}^{a_1} x_i = a_2$. Hence, the number a_2 can be thought of as a non-negative integer solution to the the above equation. In other words, the number a_2 can be thought of as a combination of a_1 integers. There are at most $\binom{a_2+a_1-1}{a_1-1} \leq 2^{2a_2}$ such combinations that add up to a_2. Similarly, the definition of the number a_3 suggests that a_3 can be thought of as a combination of a_2 integers. There are at most $\binom{a_3+a_2-1}{a_2-1} \leq 2^{2a_3}$ such combinations that add up to a_3. For positive integers x and y, let $\mathcal{C}_y^x$ denote the tuples of combinations of y into x parts. Let $\mathcal{C} = \mathcal{C}_{a_2}^{a_1} \times \mathcal{C}_{a_3}^{a_2}$.

We give a one-to-one function ϕ from $\mathcal{Z}$ to $\mathcal{C}$. Since, $|\mathcal{C}| \leq 2^{2(a_2+a_3)}$, the bound on $|\mathcal{Z}|$ follows.

Given an (a_1, a_2, a_3)-forest H, $C_1 = (|N(u_1)|, |N(u_2)|, \ldots, |N(u_{a_1}|)$ is a combination of a_1 integers that add to a_2. Let $h_i = |N(u_i)|$. For each i, we give an ordering O_i on the neighbours, say $v_1, \ldots, v_{h_i}$, of u_i such that $|N(v_1)| \leq |N(v_2)| \leq \cdots \leq |N(v_h)|$. Finally, we obtain an ordering O on the vertices of $\bigcup_{1 \leq i \leq a_1} N(u_i)$ with $O = O_1 < O_2 < \ldots < O_{a_1} = (w_1, w_2, \ldots, w_{a_2})$. That is, we order the vertices of $N(u_1)$ identical to O_1 and then the vertices of $N(u_2)$ identical to O_2 and so on. Let $C_2 = (|N(w_1) \setminus \tilde{U}_H|, |N(w_2) \setminus \tilde{U}_H|, \ldots, |N(w_{a_2}) \setminus \tilde{U}_H|)$ be a combination of a_2 integers that add to a_3. The function ϕ takes the (a_1, a_2, a_3)-forest to the pair (C_1, C_2).

Suppose there is another (a_1, a_2, a_3)-forest H' which is mapped to (C_1, C_2). Then, from the definition of ϕ, $H' \simeq H$ and hence at most one of H and H' can be part of the set of non-isomorphic (a_1, a_2, a_3)-forests.

Thus, the size of the set of non-isomorphic (a_1, a_2, a_3)-forests is upper bounded by $2^{2(a_2+a_3)}$.

We look at how to enumerate these (a_1, a_2, a_3)-forests. In fact, it is enough to enumerate all the combinations of a_1 integers that add to a_2 and a_2 integers that add to a_3. One can easily enumerate all combinations of k integers that add to n in $\mathcal{O}(\binom{n+k-1}{k-1} \cdot nk)$ time. For completeness we describe an algorithm.

Claim 6.5. *The enumeration of all combinations of k integers that add to n can be done in $\mathcal{O}(\binom{n+k-1}{k-1} \cdot nk)$ time.*

Proof. A combination of k integers that add to n is same as putting n balls into k distinct boxes. This in turn is equivalent to putting $(k-1)$ 1's in an $(n+k-1)$-bit vector. So the enumeration problem is equivalent to enumerating the set $\mathcal{V}$ of all $(n+k-1)$-bit vectors that have exactly $(k-1)$ 1's. There can be at most $\binom{n+k-1}{k-1}$ vectors in $\mathcal{V}$.

We start from the $(n+k-1)$-bit vector that has 1 in the first $k-1$ positions. For a fixed $(n+k-1)$-bit vector with $(k-1)$ 1's, let $p(i)$ denote the position of the i^{th} 1 in this vector. If $p(k-1) < n$, we generate the next vector by making $p(k-1) = p(k-1)+1$ and keeping all other positions the same. Otherwise, let $j < k-1$ be the largest integer such that $p(j+1)-p(j) > 1$. We generate the next vector by making $p(j) = p(j)+1$ and for all $i > j, p(i) = p(i)+1$. Generating a new vector takes k steps. Any two vectors that are generated differ on atleast 1 bit and every vector of $\mathcal{V}$ is generated by the algorithm. The algorithm stops with the vector where the last $k-1$ positions have 1's. The running time of the algorithm is $\binom{n+k-1}{k-1} \cdot k$

Given an $(n+k-1)$-bit vector with exactly $(k-1)$ 1's, we generate a combination of k integers $c_1, c_2, \ldots c_k$ that add to n by setting $c_1 = p(1)-1, c_k = n-p(k-1)$ and for all other i, $c_i = p(i+1)-p(i)-1$. Thus generating all combinations of k integers that add up to n takes $\mathcal{O}(\binom{n+k-1}{k-1} \cdot nk)$. □

This shows that the total number of (a_1, a_2, a_3)-forests are at most $2^{2(a_2+a_3)}$ and enumeration of this set takes at most $\mathcal{O}(2^{2(a_2+a_3)} \cdot (a_1a_2 + a_2a_3))$ time. □

Definition 6.6. *An (a_1, a_2, a_3)-pattern is constructed from an (a_1, a_2, a_3)-forest by adding a common neighbour u to each u_i, and two more vertices $\{u^1, u^2\}$ such that $\{u, u^1, u^2\}$ form a triangle. No other edges are added. See Figure 6.3 for an illustration of $(2, 4, 5)$-pattern.*

Observation 6.4. *The definition implies that there are at most $2^{2(a_2+a_3)}$ (a_1, a_2, a_3)-patterns, which can be enumerated in $\mathcal{O}(2^{2(a_2+a_3)} \cdot (a_1a_2 + a_2a_3))$ time.*

Observation 6.5. *An (a_1, a_2, a_3)-pattern without the vertices u^1, u^2 is a tree. Thus the treewidth of an (a_1, a_2, a_3)-pattern is at most 2.*

The next lemma establishes the desired connection between Gen-RBSC and Sub-Iso.

Lemma 6.9. *Let $(U = (R, B), \mathcal{F})$ be an input instance to* Gen-RBSC *parameterized by $k_\ell + d$, where each set contains at most one red element and at most d blue elements.*

Then the input instance is a YES *instance if and only if there exist integers $\hat{p}, \hat{q}, \hat{r}, \hat{s}, \hat{t}$ such that $\hat{p} \leq k_r$, $\hat{q} + \hat{s} \leq k_\ell, \hat{r} + \hat{t} = b$, H_1 is a $(\hat{p}, \hat{q}, \hat{r})$ pattern, H_2 is a $(1, \hat{s}, \hat{t})$ pattern and $(G, H = H_1 \uplus H_2)$ is a* YES *instance of* Sub-Iso.

Proof. We first show the forward direction. Since $(U = (R, B), \mathcal{F})$ is a YES instance, there exists a solution subfamily $\mathcal{F}'$ of size at most k_ℓ that covers all blue elements and at most k_r red elements. Let $\mathcal{F}_{sp} \subseteq \mathcal{F}'$ be such that for all $F \in \mathcal{F}_{sp}, F \cap R = \emptyset$ and let $\mathcal{F}'' = \mathcal{F}' \setminus \mathcal{F}_{sp}$. Let $\hat{q} = |\mathcal{F}''|$, $\hat{s} = |\mathcal{F}_{sp}|$ and let $\hat{p} \leq k_r$ denote the number of red elements covered by $\mathcal{F}''$. Also, let $\hat{r} \leq b$ be the number of blue points covered by $\mathcal{F}''$ and $\hat{t} \leq b$ be $b - \hat{r}$. We show that there exists $H = H_1 \uplus H_2$ such that H_1 is a $(\hat{p}, \hat{q}, \hat{r})$ pattern and H_2 is a $(1, \hat{s}, \hat{t})$ pattern and (G, H) is a YES instance of Sub-Iso.

- We define an arbitrary ordering $I_R = \{r_1, r_2, \ldots r_{\hat{p}}\}$ on the red elements covered by $\mathcal{F}'$. Let $\mathcal{F}'_i$ denote the family of subsets of $\mathcal{F}'$ that contains the red element r_i.

  For, $1 \leq i \leq \hat{p}$, let $|\mathcal{F}'_i| = p^1_i$. This gives us a $\hat{p}$ partition $P^1 = (p^1_1, p^1_2, \ldots, p^1_{\hat{p}})$ of $\hat{q}$.

- For every $i \leq \hat{p}$, we give an arbitrary ordering I_i to the sets in $\mathcal{F}'_i$.

  Finally, we consider the ordering $I = I_1 < I_2 < \ldots < I_{\hat{p}} = (S_1, S_2, \ldots, S_{\hat{q}})$ to the sets in $\mathcal{F}'$. That is, we first order the vertices of $\mathcal{F}'_1$ identical to I_1 and then the vertices of $\mathcal{F}'_2$ identical to I_2 and so on.

- We pick each set in $\mathcal{F}'$ in the order given by I. We assign to that set the number of blue elements, contained in the set, that are not contained in sets of smaller index. In particular, let $B(S_i)$ denote the set of blue elements contained in S_i and not contained in any S_j with $j < i$. Then to each set S_i we assign an integer $p^2_{S_i} = |B(S_i)|$.

  This gives us a $\hat{q}$-partition $P^2 = (p^2_{S_1}, p^2_{S_2}, \ldots, p^2_{S_{\hat{q}}})$ of $\hat{r}$.

- We create $\hat{p}$ components where, for component i, u_i corresponds to the subfamily of sets $\mathcal{F}'_i$. In fact, for $i < \hat{p}$, u_i corresponds to the red element r_i. For each u_i we create p^1_i neighbours, say, $w^i_1, \ldots, w^i_{p^1_i}$. Clearly, $\Sigma_{1 \leq i \leq \hat{p}} |N(u_i)| = \hat{q}$.

- We identify the vertex w^i_s with a set in $\mathcal{F}'_i$. In particular, we assign w^i_s to S_t where $t = s + \Sigma_{1 \leq j \leq i-1} p^1_j$. For each vertex w^i_s we add $p^2_{S_t}$ number of children. Clearly, by construction $\Sigma_{1 \leq i \leq \hat{p}} |N(N(u_i)) \setminus \{u_i\}| = \hat{r}$.

We have constructed a $(\hat{p}, \hat{q}, \hat{r})$-forest now. Similarly we construct a $(1, \hat{s}, \hat{t})$-forest with $u_{\hat{p}+1}$ corresponding to the sets in $\mathcal{F}''$. $u_{\hat{p}+1}$ has $\hat{s}$ neighbours, $w^{\hat{p}+1}_1, \ldots, w^{\hat{p}+1}_{\hat{s}}$, corresponding to $\hat{s}$ sets in $\mathcal{F}''$ and for each such set S_i, the corresponding $w^{\hat{p}+1}_i$ has $p^2_{S_i} = |B(S_i)|$ children.

Now, we construct the corresponding pattern graphs H_1 and H_2 from these forests by introducing some new vertices(By Definition 6.6). All the vertices $u_1, \ldots, u_{\hat{p}}$ are connected to a new vertex u which forms a triangle with u^1 and u^2. Similarly, $u_{\hat{p}+1}$ is connected to a new vertex u' which forms a triangle with u^3 and u^4. Finally, we show that (G, H) is a YES instance of Sub-Iso. Towards this we define an isomorphism function f in the following way.

1. $f(u) = v_R, f(u^1) = v_1, f(u^2) = v_2$. $f(u') = v_N, f(u^3) = v_3, f(u^4) = v_4$
2. For all $1 \leq i \leq \hat{p}$, $f(u_i) = v_{r_i}$ and $f(u_{\hat{p}+1}) = v_{sp}$.
3. For all $1 \leq i \leq \hat{p}+1$ and $1 \leq s \leq p^1_i$, we map $f(w^i_s) = v_{S_t}$ where $t = s + \Sigma_{1 \leq j \leq i-1} p^1_j$.
4. Recall that for each vertex w^i_s we added $p^2_{S_t}$ number of children. To these children we arbitrarily assign a unique blue element contained in $B(S_t)$.

It is easy to see that f defines a graph isomorphism from H to a subgraph of G. Thus, (G, H) is a YES instance of Sub-Iso.

Now assume there is a H such that $H = H_1 \uplus H_2$ where H_1 is a $(\hat{p}, \hat{q}, \hat{r})$ pattern and H_2 is a $(1, \hat{s}, \hat{t})$ pattern. From the definition of pattern graphs and the construction of G, it follows that f maps the vertices u, u^1, u^2 in H_1 (resp. u', u^3, u^4 in H_2) to v_R, v_1, v_2 (resp. v_N, v_3, v_4) respectively.

This implies that, $f(\{u_1, \ldots, u_{\hat{p}}\}) \subseteq V_R$, $f(u_{\hat{p}+1}) = v_{sp}$, $f\Big(\bigcup_{1 \leq i \leq (\hat{p}+1)} N(u_i)\Big) \subseteq V_{\mathcal{F}}$ and $f\Big(\bigcup_{1 \leq i \leq (\hat{p}+1)} (N(N(u_i)) \setminus \{u_i\})\Big) \subseteq V_B$. This means that the family of sets corresponding

to the vertex set $f(\bigcup_{1\leq i\leq(\hat{p}+1)} N(u_i))$ covers at most k_r red elements and all blue elements. Hence, the input instance is a YES instance. This completes the proof. □

For our main algorithm we also need the following known algorithm for Sub-Iso.

Proposition 6.2 ([Fomin 2014a, Fomin 2014b])**.** *Let G and H be two graphs on n and q vertices respectively and the treewidth of H is at most t. Then, there is a deterministic algorithm for* Sub-Iso *that runs in time* $2.619^q(nt)^{t+\mathcal{O}(1)}$.

Using all the above details, we arrive at an FPT algorithm for this case of Gen-RBSC.

Theorem 6.16. *The* Gen-RBSC *problem parameterized by $k_\ell + d$, where each set in the input instance contains at most* 1 *red element and at most d blue elements, is* FPT*. The running time of the* FPT *algorithm is* $2^{\mathcal{O}(dk_\ell)} \cdot (|U| + |\mathcal{F}|)^{\mathcal{O}(1)}$.

Proof. By assumption, every set can have at most d blue elements and at most 1 red element. Thus, a YES instance has $b \leq dk_\ell$. Also, by the assumption on the input instance, the solution sets can contain at most k_ℓ red elements. If $k_r \leq k_\ell$, then a solution family will have at most k_r red elements, otherwise it will have at most k_ℓ red elements by definition of the problem. So, it is safe to assume that $k_r \leq k_\ell$.

Given an input of Gen-RBSC we create the host graph G as described above. Lemma 6.9 gives us a characterization for the YES instances, in terms of this host graph and (a_1, a_2, a_3)-pattern graphs. That is, the input instance is a YES instance if and only if there exist integers $\hat{p}, \hat{q}, \hat{r}, \hat{s}, \hat{t}$ such that $\hat{p} \leq k_r$, $\hat{q} + \hat{s} \leq k_\ell, \hat{r} + \hat{t} = b$ and H_1 is a $(\hat{p}, \hat{q}, \hat{r})$ pattern and H_2 is a $(1, \hat{s}, \hat{t})$ pattern and $(G, H = H_1 \uplus H_2)$ is a YES instance of Sub-Iso. We use this characterization to obtain our algorithm. We enumerate all patterns with the given conditions true, using Lemmata 6.8 and Observation 6.4. Using bounds on binomial expansions and geometric series sums, the number of such patterns, is $2^{\mathcal{O}(dk_\ell)}$.

Since removing one vertex each from H_1 and H_2 makes H a forest, the tree width of H is 2. For each such pattern $\tilde{H}$ we check whether $(G, \tilde{H})$ is a YES instance of Sub-Iso using Proposition 6.2. If, for any pattern $\tilde{H}$, $(G, \tilde{H})$ is a YES instance of Sub-Iso then we return YES. Otherwise, we return NO. The correctness of this step follows from Lemma 6.9. The running time of this algorithm is $2^{\mathcal{O}(dk_\ell)} \cdot (|U| + |\mathcal{F}|)^{\mathcal{O}(1)}$. This concludes the proof. □

A Dynamic Programming Algorithm

We give a Dynamic Programming algorithm to solve Gen-RBSC parameterized by $k_l + d$, for the case when all sets contain at most 1 red element and at most d blue elements. Our algorithm guesses the red point that can be added to the solution one by one and also guesses the sets that can cover it and covers the remaining blue points optimally.

Lemma 6.10. *There exists a* FPT *algorithm that solves* Gen-RBSC *when each set in the input instance contains at most* 1 *red element and at most d blue elements. The running time of this algorithm is* $O(2^{2dk_l}(|U| + |\mathcal{F}|)^{\mathcal{O}(1)})$.

Proof. Let $B' \subseteq B$, $r' \in R \cup nil, j \in \mathbb{N}$. Let $W[B', r']$ represent the minimum cardinality of a family $\mathcal{F}' \subseteq \mathcal{F}$ that covers all elements in B' and does not cover any red element

except r' (no red element if r' is nil). The value of $W[B', r']$ is $+\infty$ if no such $\mathcal{F}' \subseteq \mathcal{F}$ exists. Let $T[B', j]$ represent the minimum cardinality of a family $\mathcal{F}' \subseteq \mathcal{F}$ that covers all elements in B' and covers at most j red elements. Clearly the instance is a YES instance if and only if $T[B, k_r] \leq k_l$.

We can compute the value of $T[B, k_r]$ using the following recurrence.
$T[B', 0] = W[B', nil]$
$T[B', j] = \min_{r' \in (R \cup nil)} \min_{B'' \subseteq B'} (W[B'', r'] + T[B' \setminus B'', j-1])$

Similarly we can compute the value of $W[B', r']$ using the following recurrence.
$W[\emptyset, r'] = 0$
$W[B', r'] = 1 + \min_{S \in \mathcal{F}, S \cap R = \emptyset \text{ or } S \cap R = \{r'\}, S \cap B' \neq \emptyset} W[B' \setminus S, r']$
Let us first show that the recurrence for W is correct. The proof is by induction on $|B|$. When $|B| = 0$ the recurrence correctly returns zero. When $|B| > 0$, $W[B' \setminus S, r']$ returns the minimum cardinality of a family $\mathcal{F}' \subseteq \mathcal{F}$ that covers all elements in $B' \setminus \mathcal{F}$ and does not cover any red element except r' (by induction hypothesis). Therefore, $S \cup \mathcal{F}'$ covers all elements in B' and does not cover any red element except r'. Since we are doing this for every $S \in \mathcal{F}$ and take the minimum value, the recurrence indeed returns the minimum cardinality of a family $\mathcal{F}' \subseteq \mathcal{F}$ that covers all elements in B' and does not cover any red element except r'.

Now we show that the recurrence for T is correct by induction on j. When $j = 0$, the recurrence returns the value of $W[B, nil]$ which returns the minimum cardinality of a family $\mathcal{F}' \subseteq \mathcal{F}$ that covers all elements in B' and does not cover any red element. When $j > 0$, we consider a number of sets containing the same red element r', paying for the blue elements $B'' \subseteq B'$ they cover, and cover the remaining blue elements $B' \setminus B''$ optimally by induction hypothesis. Since we do this for all red points and return the minimum value, the recurrence is correct.

Running time: To compute the value of $T[B, k_r]$ using the above recurrence, we have to compute at most $2^{|B|}|U|$ values of W and T, which is at most $2^{dk_l}|U|$ in YES-instances. Every value of W can be computed in $O(|U|)$ time using previously computed values. To compute a value of T, we take the minimum over all choices of r' in R, over at most $2^{|B|} \leq 2^{dk_l}$ choices of B'', and look up earlier values. Thus the running time is bounded by $O(2^{2dk_l}(|U| + |\mathcal{F}|)^{\mathcal{O}(1)})$. □

When it comes to kernelization for this special case, we show that even for Gen-RBSC-lines parameterized by $k_\ell + d$ there cannot be a polynomial kernel unless CoNP $\subseteq$ NP/poly. For this we will give a polynomial parameter transformation from Set Cover parameterized by universe size n. The ppt reduction is exactly the one given in Theorem 6.8.

Theorem 6.17. Gen-RBSC-lines *parameterized by $k_\ell + d$, and where every line has at most* 1 *red element and at most d blue elements, does not allow a polynomial kernel unless* CoNP $\subseteq$ NP/*poly.*

6.11 Chapter Summary

In this chapter, we provided a complete parameterized and kernelization dichotomy of the Gen-RBSC-lines problem, under all possible combinations of its natural parameters. We also studied RBSC-lines and Gen-RBSC under different parameterizations. The next natural step seems to be a study of the Gen-RBSC problem, when the sets are hyperplanes. Another interesting variant is when the set system has bounded intersection.

We believe that the running time of the FPT algorithm for Gen-RBSC-lines parameterized by k_ℓ, k_r is tight, up to the constants appearing in the exponents. It would be interesting to show that the problems cannot have algorithms with running time dependence on parameters as $k_\ell^{o(k_\ell)} \cdot k_r^{\mathcal{O}(k_r)}$ or $k_\ell^{\mathcal{O}(k_\ell)} \cdot k_r^{o(k_r)}$, under standard complexity theoretic assumptions (like the Exponential Time Hypothesis).

Chapter 7

Unique Covering problems with Geometric Sets

7.1 Introduction

In this chapter, we continue to study the parameterized complexity of variants of Set Cover. The classic Set Cover problem is the following: For a set system $(U, \mathcal{F})$ where U is a finite universe of n elements and $\mathcal{F}$ is a family of subsets of U, is there a sub family of at most k sets in $\mathcal{F}$ whose union is U? We say that an element x in U is *covered* by a set S from $\mathcal{F}$ if the set S contains the element x.

For several applications, it turns out that we would like to not only cover elements of U using sets in $\mathcal{F}$, but also cover them uniquely. A common motivation involves problems where covering elements by more than one set leads to noise (for example, wireless networks), so we would like to ensure that an element is covered, but by only one of the sets. This desired refinement manifests itself in the following three natural variations of the Set Cover problem.

- All elements must be covered uniquely by at most k sets. (Exact Cover)
- All elements must be covered, *and* at least k elements must be covered uniquely. (Unique Set Cover)
- At least k elements are covered uniquely. (Unique Cover)

In the first two variants, we are looking for a set cover with additional properties. Note that in the last setting, a valid solution may not be a set cover.

The Exact Cover problem was one of the twenty-one problems shown to be NP-complete by Karp [Karp 1972]. The Unique Cover problem was introduced by Demaine et al in [Demaine 2008], and it may be considered a natural "maximization" variant of Set Cover, and also a generalisation of the Max Cut problem. The Unique Set Cover problem combines elements of both these variants, and is NP-complete as well.

The Geometric Setting. Geometric settings are among the most promising contexts for developing improved algorithms when faced with hardness in a general setting. The geometric nature of the problem opens up several algorithmic possibilities, and this is amply evidenced in the context of approximation algorithms. Many geometric problems are known to admit good approximation algorithms, even PTASes. In particular, the classical set cover and hitting set problems have been very well-explored in the context

	Exact Cover		Unique Cover	Unique Set Cover
Parameter	Size of solution		Number of elements uniquely covered	
Abstract Sets	W[1]-hard		FPT	W[1]-hard
			Quadratic Element Kernel	
Lines	FPT	VC-Dimension	FPT	FPT
	Quadratic Kernel		Quadratic Kernel	Poly Kernel
Hyperplanes $\mathbb{R}^d$	$k^{O(d^2)}$ kernel		$k^{O(d)}$ Instance Kernel (Quadratic Element Kernel)	$k^{O(d^2)}$ kernel
Unit Squares	W[1]-hard		Poly Kernel	Open

Table 7.1: A summary of our results.

of geometric objects [Hochbaum 1987, Mustafa 2009]. In this situation, the universe is a point set in d-dimensional Euclidean space, and the sets are defined by intersection of geometric objects with the point set. An object covers a point if it contains it. We study the unique coverage variants for several geometric objects, including lines, hyperplanes, squares, and rectangles.

Our Approach. We focus on the parameterized complexity of these problems, both in the abstract setting and in carefully considered special cases. Studying the parameterized complexity of geometric problems has interesting implications. On the one hand, a tractability result demonstrates the utility of the geometric structure in contrast with the abstract setting. On the other, a hardness result often has consequences for hardness of approximation; usually it establishes evidence for the non-existence of the EPTAS. This has motivated several studies of geometric problems from a parameterized perspective [Marx 2005].

Our Results. This work is based on the results obtained in [Ashok 2015b]. We establish the following results, summarised also in Table 7.1.

Exact Cover We show that Exact Cover is W[1]-hard even in the restricted setting where all the objects are unit squares (Lemma 7.1). On the positive side, we show that Exact Cover is FPT for lines (Lemma 7.2). Further, if the objects are hyperplanes in a d-dimensional Euclidean space, the Exact Cover continues to be FPT parameterized by k and d (Lemma 7.3).

Unique Cover For Unique Cover, a simple argument shows that the number of elements in the universe can be bounded by $O(k^2)$ (Lemma 7.4). This shows that the problem is FPT. It turns out that this also implies a polynomial kernel for various geometric objects (Corollary 7.2), using the fact that these objects have bounded VC Dimension.

Unique Set Cover We show that Unique Set Cover is W[1]-hard in the general setting (Lemma 7.5) and NP-complete when restricted to lines (Lemma 7.6). On the positive side, we show that the problem is FPT for families of bounded intersection (Lemma 7.7) and hyperplanes in d dimensions (Lemma 7.8).

7.2 Preliminaries

Problem Definitions A set system is a pair $(U, \mathcal{F})$, where U is a universe of n elements and $\mathcal{F}$ is a family of m subsets of U. Given a set system $(U, \mathcal{F})$, a set S is said to cover an element $p \in U$ if $p \in S$. An element p is said to be covered *uniquely* by a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ if there is exactly one set in $\mathcal{F}'$ which contains p. The Set Cover problem asks for a smallest collection of subsets whose union covers every element in the universe. We are now ready to define some of the variations of this problem that we consider in our work.

Exact Cover	**Parameter:** k
Input: A set system $(U, \mathcal{F})$ of n elements and m sets, and a positive integer k.	
Question: Is there a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ of size at most k that covers every element of U such that each element is contained in exactly one set in $\mathcal{F}'$?	

Unique Cover	**Parameter:** k
Input: A set system $(U, \mathcal{F})$ of n elements and m sets, and a positive integer k.	
Question: Is there a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ such that, among the set of elements covered by $\mathcal{F}'$, there is a subset $S \subset U$, $\lvert S\rvert \geq k$ with each element of S being contained in exactly one set in $\mathcal{F}'$?	

Unique Set Cover	**Parameter:** k
Input: A set system $(U, \mathcal{F})$ of n elements and m sets, and a positive integer k.	
Question: Is there a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ that covers every element of U such that there is a subset $S \subseteq U$, $\lvert S\rvert \geq k$ where each element of S is contained in exactly one set in $\mathcal{F}'$?	

We consider some notions that will be useful in defining special set systems that we will encounter during our study of these problems. A set system $(U, \mathcal{F})$ is said to have *bounded intersection* when there is a universal constant c such that for any pair of sets $F_1, F_2 \in \mathcal{F}$, $|F_1 \cap F_2| \leq c$. A set system $(U, \mathcal{F})$ is said to be a set system of squares (lines) if U is a subset of n points in $\mathbb{R}^2$ and each set $F \in \mathcal{F}$ is the maximal set of points of U that are contained in a square(line) defined on $\mathbb{R}^2$. Similarly, a set system $(U, \mathcal{F})$ is said to be a set system of hyperplanes if U is a set of n points in $\mathbb{R}^d$, for a fixed positive integer d, and each set $F \in \mathcal{F}$ is the maximal set of points of U that are contained in a hyperplane defined on $\mathbb{R}^d$.

Given a set system $(U, \mathcal{F})$ of n elements and m sets, for every subset $A \subseteq U$ we define the family of sets $\mathcal{F}_A = \{S \cap A | S \in \mathcal{F}\}$.

Definition 7.1 (VC Dimension)**.** *Let $(U, \mathcal{F})$ represent a set system. A subset $A \subseteq U$ is said to be* shattered *if for every $B \subseteq A$, there exists $F \in \mathcal{F}$ such that $F \cap A = B$. The* Vapnik-Chervonenkis dimension *(or* VC dimension*) of $(U, \mathcal{F})$ is the supremum of the sizes of all shattered subsets of U.*

Therefore, in general, the VC dimension of a set system could be infinite. However, set systems of several geometric objects are known to have bounded VC dimension. We refer the reader to [Matoušek 2002] for further details on VC Dimension. The following result is known from [Vapnik 1971] about set systems of finite VC dimension.

Proposition 7.1. *Let $(U, \mathcal{F})$ be a set system with $|U| = n$ and VC dimension d. Then $|\mathcal{F}| \leq \binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{d}$*

Hyperplanes An i-flat in $\mathbb{R}^d$ is the affine hull of $i+1$ affinely independent points. The dimension of a (possibly infinite) set of points P, denoted as $dim(P)$, is the minimum i such that the entire set P is contained in an i-flat of $\mathbb{R}^d$ [Langerman 2005].

Observation 7.1 ([Langerman 2005])**.** *For a pair of i-flat H_1 and j-flat H_2 , $1 \leq j, i \leq d-1$, if $H_1 \not\subset H_2$ and $H_2 \not\subset H_1$, then $dim(H_1 \cap H_2) < min\{i, j\}$.*

We refer to $(d-1)$-flats of $\mathbb{R}^d$ as hyperplanes.

7.3 Exact Cover

In this section, we consider the Exact Cover problem, parameterized by the number k of sets in a solution family. Since this problem is known to be W[1]-hard, it is natural to introduce properties on the input set family and see whether the added structure makes the problem easier in these special cases. Here, we restrict ourselves to geometric versions, where the universe is a set of points in a real space $\mathbb{R}^d$, for an appropriate integer d, while the set family is such that every set satisfies a particular geometric property.

The W[1]-hardness of Exact Cover was shown in [Misra 2013]. We give an alternative proof for this W[1]-hardness. This proof, with a little bit of modification, shows that Exact Cover on set systems of unit squares is also W[1]-hard.

Proposition 7.2. Exact Cover *is* W[1]*-hard.*

Proof. We give a polynomial time reduction from the Clique problem, which is known to be W[1]-hard when parameterized by the size k of the solution clique [Downey 2012]. We begin by giving a reduction from the Exact Cover in the abstract setting. Given an instance (G, k) of the Clique problem, we construct the following instance $(U, \mathcal{F}, k')$ of Exact Cover. For the ease of description, we will assume that if a set $[a, b]$ is such that $a > b$ then this set is equivalent to the empty set. In particular, a set $[n+1, n]$ is an empty set. The construction is as below:

- We add k^2 new elements $U' = \{e_{a,b} | 1 \leq a, b \leq k\}$.
- We create $k(k-1)$ copies of $V(G) = \{1, 2, \ldots, n\}$ denoted as $H_{a,b}, a \leq k, b \leq k-1$. We also create $k(k-1)$ copies of $V(G)$ denoted as $V_{a,b}, a \leq k-1, b \leq k$. $U = U' \cup \bigcup_{a,b}(H_{a,b} \cup V_{a,b})$.
- For each $i \in V(G)$, we construct a set $S^{1,1}_{i,i} = H_{1,1}[1, \ldots, i] \cup V_{1,1}[1, \ldots, i] \cup \{e_{1,1}\}$ and a set $S^{k,k}_{i,i} = H_{k,k-1}[i+1, \ldots, n] \cup V_{k-1,k}[i+1, \ldots, n] \cup \{e_{k,k}\}$.
- For every edge $ij \in E(G)$, we construct a set $S^{k,1}_{i,j} = H_{k,1}[1, \ldots, i] \cup V_{k-1,1}[j+1, \ldots, n] \cup \{e_{k,1}\}$ and a set $S^{1,k}_{i,j} = H_{1,k-1}[i+1, \ldots, n] \cup V_{1,k}[1, \ldots, j] \cup \{e_{1,k}\}$.
- For each $1 < a < k$, and every vertex $i \in V(G)$, we construct a set $S^{a,a}_{i,i} = H_{a,a-1}[i+1, \ldots, n] \cup H_{a,a}[1, \ldots, i] \cup V_{a-1,a}[i+1, \ldots, n] \cup V_{a,a}[1, \ldots, i] \cup \{e_{a,a}\}$.

- For each $1 < a < k$, and every edge $ij \in E(G)$, we construct a set $S^{a,1}_{i,j} = H_{a,1}[1,\ldots,i] \cup V_{a-1,1}[j+1,\ldots,n] \cup V_{a,1}[1,\ldots,j] \cup \{e_{a,1}\}$ and a set $S^{a,k}_{i,j} = H_{a,k-1}[i+1,\ldots,n] \cup V_{a-1,k}[j+1,\ldots,n] \cup V_{a,k}[1,\ldots,j] \cup \{e_{a,k}\}$.

- For each $1 < b < k$, and every edge $ij \in E(G)$, we construct a set $S^{1,b}_{i,j} = H_{1,b-1}[i+1,\ldots,n] \cup H_{1,b}[1,\ldots,i] \cup V_{1,b}[1,\ldots,j] \cup \{e_{1,b}\}$ and a set $S^{k,b}_{i,j} = H_{k,b-1}[i+1,\ldots,n] \cup H_{k,b}[1,\ldots,i] \cup V_{k-1,b}[1,\ldots,j] \cup \{e_{k,b}\}$.

- For each $1 < a \neq b < k$, and every edge $ij \in E(G)$, we construct a set $S^{a,b}_{i,j} = H_{a,b-1}[i+1,\ldots,n] \cup H_{a,b}[1,\ldots,i] \cup V_{a-1,b}[j+1,\ldots,n] \cup V_{a,b}[1,\ldots,j] \cup \{e_{a,b}\}$.

- $\mathcal{F} = \{S^{a,a}_{i,i} | i \in V(G), 1 \leq a \leq k\} \cup \{S^{a,b}_{i,j} | ij \in E(G), 1 \leq a \neq b \leq k\}$.

- $k' = k^2$.

In the construction, for an element $e_{a,a}, 1 \leq a \leq k$, we ensure that each set containing $e_{a,a}$ encodes a distinct vertex of the graph G. On the other hand, for an element $e_{a,b}, 1 \leq a \neq b \leq k$, each set containing $e_{a,b}$ encodes a distinct edge of G.

Claim 7.1. *The constructed* Exact Cover *instance,* $(U, \mathcal{F}, k')$*, has a solution of size* $k' = k^2$ *if and only if there is a clique of size* k *in* G*.*

Proof. Assume there is a clique C of size k in G. Let $V(C) = \{i_1 > i_2 > \ldots > i_k\}$. Then the sets $\{S^{j,j}_{i_j,i_j} | 1 \leq j \leq k\} \cup \{S^{j,l}_{i_j,i_l} | 1 \leq j \neq l \leq k\}$ form a solution for the above Exact Cover instance.

Conversely, let $\mathcal{S}$ be a solution for the Exact Cover instance. For any $(a,b), 1 \leq a,b \leq k$, two sets $S^{a,b}_{i_1,j_1}$ and $S^{a,b}_{i_2,j_2}$ will have the point $e_{a,b}$ in their intersection. Therefore, for a pair (a,b) atmost one set $S^{a,b}_{i,j}$ can be picked in any solution. In fact, the point $e_{a,b}$ can only be covered by a set $S^{a,b}_{i,j}, 1 \leq i,j \leq n$. This implies that for every pair (a,b), exactly one set $S^{a,b}_{i,j}$ belongs to $\mathcal{S}$. Let $\{i_1, i_2, \ldots, i_k\}$ be indices such that $\{S^{j,j}_{i_j,i_j} | 1 \leq j \leq k\} \subseteq \mathcal{S}$. The construction ensures that any other set picked in $\mathcal{S}$ corresponds to an edge of G. For a pair (a,b), if a set $S^{a,b}_{i,j}$ is picked with $i < n$, then to cover the element $H_{a,b}[i+1]$ while maintaining exact coverage, we must pick a set $S^{a,b+1}_{i,l}$, for some $1 \leq l \leq k$. When $i = n$, then to cover the element $e_{a,b+1}$ while maintaining unique coverage, we must pick a set $S^{a,b+1}_{i,l}$, for some $1 \leq l \leq k$. By similar arguments, if for a pair (a,b), a set $S^{a,b}_{i,j}$ is picked, we must pick a set $S^{a+1,b}_{l,j}$ where $1 \leq l \leq k$. This implies that for a pair $1 \leq a \neq b \leq k$, the set chosen in $\mathcal{S}$ must be $S^{a,b}_{i_a,i_b}$, where $S^{a,a}_{i_a,i_a}$ and $S^{b,b}_{i_b,i_b}$ have been chosen respectively to cover $e_{a,a}$ and $e_{b,b}$. Then, by construction, $i_a \neq i_b$ and $i_a i_b \in E(G)$ in the graph G. Thus, the set $\{i_1, i_2, \ldots, i_k\}$ corresponds to a set of k distinct vertices in G, which are pairwise adjacent. This tells us that the induced subgraph $G[i_1, \ldots, i_k]$ is a clique in G. □

This proves W[1]-hardness for Exact Cover. □

We can also show the hardness of Exact Cover on set systems of unit squares.

Lemma 7.1. Exact Cover *on set systems of unit squares is* W[1]*-hard.*

Proof. We will show that the above reduction in the general case can in fact be made to work for the special case, when each set is a unit square. This will complete the proof of this lemma. We need the following specifications for the reduction to go through for this case (See Figure 7.3 and Figure 7.3):

- The element $e_{a,b} \in U'$ is the point (b, a).
- The element $H_{a,b}[i], 1 \leq i \leq n$, is the point $(b + \frac{i}{n+1}, a)$.
- The element $V_{a,b}[i], 1 \leq i \leq n$, is the point $(b, a + \frac{i}{n+1})$.
- The set $S_{i,j}^{a,b}, 1 \leq i, j \leq n, 1 \leq a, b \leq k$, is the unit square defined by the 4 lines $x = (b-1) + \frac{i}{n+1} + \frac{1}{2(n+1)}, x = b + \frac{i}{n+1} + \frac{1}{2(n+1)}, y = (a-1) + \frac{j}{n+1} + \frac{1}{2(n+1)}, y = a + \frac{j}{n+1} + \frac{1}{2(n+1)}$.

Other than this, the construction of the instance of Exact Cover from an instance (G, k) of the Clique problem is the same as in Proposition 7.2. The argument for the correctness of the reduction is also the same as above. This proves that Exact Cover for unit squares is W[1]-hard. □

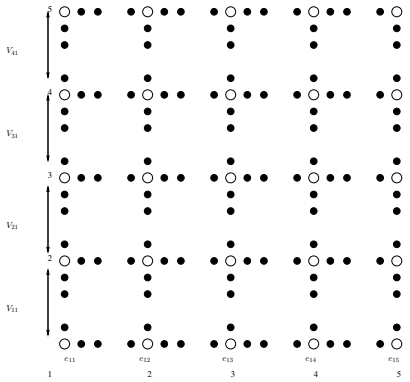


Figure 7.1: Arrangement of points in a reduced instance.

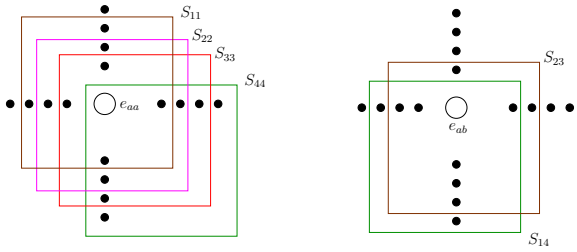


Figure 7.2: Sets in a reduced instance.

Kernels for Exact Cover with Lines and Hyperplanes. In contrast with the hardness results that we have seen so far, we now turn to some algorithmic results. First, when we consider our input universe to be a set of n points in $\mathbb{R}^2$ and our sets to be maximal sets of collinear points, we obtain a quadratic kernel using a "high degree" reduction rule.

This version of Exact Cover is also NP-complete, and the proof for NP-hardness is very similar to the proof of Lemma 7.6.

Let $(P, \mathcal{L})$ be the input set system. In our discussion, we use the terms sets and lines interchangeably. The input family could contain sets containing single points. Our first reduction rule is taken directly from [Langerman 2005]. We remove all lines (but for one) that pass through exactly one point.

Reduction Rule 7.1. *For any input point p, from the set $\mathcal{L}_p = \{L | L \in \mathcal{L}$ and $L \cap P = \{p\}\}$, delete all lines but one (say L_p).*

Proposition 7.3. *Reduction Rule 7.1 is sound.*

Proof. Suppose $\mathcal{L}'$ is a solution for the given Exact Cover instance. Clearly, atmost one set L from the set $\mathcal{L}_p$ can belong to $\mathcal{L}'$. If $\mathcal{L}'$ does not contain any lines from $\mathcal{L}_p$, or contains $\mathcal{L}_p$, there is nothing to prove. Otherwise, let $L \in \mathcal{L}_p$ belong to $\mathcal{L}'$. Observe that $(\mathcal{L}' \setminus L) \cup L_p$ is also a valid exact cover of the same size at $\mathcal{L}'$, and this proves the correctness of the reduction rule. □

Reduction Rule 7.2. *In an instance $(P, \mathcal{L}, k)$, if there is a line L that contains at least $k+1$ input points then delete L and all lines intersecting with points on L and decrease k by one. All points contained in L are deleted from the universe U. Formally, the reduced instance is $(P \setminus L, \mathcal{L} \setminus \{T \mid L \cap T \neq \emptyset\}, k-1)$.*

A similar reduction was given in [Langerman 2005] to exhibit a polynomial kernel for Point Line Cover. We show the correctness of this reduction in this case.

Claim 7.2. *Reduction Rule 7.2 is sound.*

Proof. Suppose there is a solution, $\mathcal{L}'$, for $(P, \mathcal{L}, k)$ which does not contain L. Since $\mathcal{L}'$ is a set cover of size atmost k that excludes L, it includes at least $(k+1)$ other lines to cover the points on L, as two lines intersect at atmost one point. This contradicts that $\mathcal{L}'$ is a solution of the instance $(P, \mathcal{L}, k)$. Note that this argument shows that *any* valid solution of the instance $(P, \mathcal{L}, k)$ must contain L. Now, consider the subfamily of $\mathcal{L}$ comprising of lines that contain points belonging to L:

$$\mathcal{L}'' := \{T \in \mathcal{L} \mid T \neq L, T \cap L \neq \emptyset\}.$$

Clearly, any solution that contains L does not contain any element of $\mathcal{L}''$, therefore, it is safe to remove these sets from the instance, establishing the correctness of Reduction Rule 7.2. □

The reduction is very robust, in the sense that a line as described above will belong to any solution of the Exact Cover instance. On exhaustive application of this reduction, a YES instance can have atmost k^2 remaining input points, since k lines can only cover k^2 points when each line has atmost k points.

Therefore, if our reduced instance has more than k^2 points, we correctly return NO. Otherwise, due to Reduction Rule 7.1 and by the property of lines, we can also bound the number of lines in the reduced instance to atmost k^4. Thus, we have shown the following.

Lemma 7.2. Exact Cover *on set systems of lines is* FPT*, with a polynomial kernel.*

A natural question is to consider input instances of Exact Cover which are set systems of hyperplanes in $\mathbb{R}^d$. We parameterize the Exact Cover problem in this case by $k+d$. The following Lemma is obtained by extending the reduction rules in [Langerman 2005]. In particular, it is shown in [Langerman 2005] that for any $1 \leq i \leq d-1$, if an i-flat covers more than $k^i + 1$ points, then these points can be replaced with one representative. The crux of the argument is that all of these points are covered "together" by a single hyperplane in any valid solution. In our argument, we further this reduction rule by deleting all hyperplanes that contain a strict subset of these points, because such hyperplanes are automatically forbidden from being a part of any valid solution of Exact Cover.

Lemma 7.3. Exact Cover *on set systems of hyperplanes in* $\mathbb{R}^d$ *is* FPT *parameterized by* $k+d$.

Proof. First, we try to reduce the number of elements per hyperplane.

Reduction Rule 7.3. *Repeat for* $i = 1$ *to* $d-2$*:*
If there exists $P' \subset P$ *such that* $|P'| \geq k^i + 1$ *and all points in* P' *lie on a* i*-flat, then delete all but one points from* P'*. We denote the single point that remains behind as* p'*. Also, delete all hyperplanes in* $\mathcal{F}$ *which intersect with a proper subset of* P'*.*

The correctness of replacing each set P' by a point p is similar to what is shown in [Langerman 2005]. By deleting all hyperplanes in $\mathcal{F}$, that intersect partially with P', we ensure that a hyperplane covers P' of the original instance if and only if it covers p' of the reduced instance. It is easy to see that an Exact Cover solution for the original instance is also an Exact Cover solution for the reduced instance. On the other hand, in the original instance, if there is a set cover of size atmost k, then all the points of P' must be covered together. This is represented by the point $p' \in P'$. This ensures that an Exact Cover solution for the reduced instance is an Exact Cover solution for the original instance. After exhaustive application of this Reduction Rule, [Langerman 2005] showed that any hyperplane H with at least $k^{(d-1)} + 1$ points must be contained in any set cover of size atmost k. In particular, H must be covered in any Exact Cover solution of size atmost k. Also, any hyperplane intersecting with H cannot be included in any solution of Exact Cover, to maintain unique coverage of each point of U. Thus, all hyperplanes intersecting with H can be deleted from our instance. Also, the points covered by H will not be covered again and hence can be deleted from our instance. A formal argument is very similar to the proof of 7.2. The Reduction Rule, stated formally, is given below:

Reduction Rule 7.4. *Let* H *be a hyperplane that contained at least* $k^{d-1}+1$ *points of* U*. Then this hyperplane must be included in any* Exact Cover *solution. All hyperplanes intersecting with* H*, along with* H *itself are deleted. The points contained in* H *are deleted. The budget* k *is reduced by* 1*.*

After exhaustive application of this Reduction Rule, each hyperplane can contain atmost k^{d-1} points. Since an Exact Cover solution is also a set cover of size atmost k, there can be atmost k^d points in the reduced instance. The VC dimension of a hyperplane in $\mathbb{R}^d$ is d. From Proposition 7.1, for a set system of VC Dimension d, the number of distinct sets is bounded by $O(n^d)$. Hence, the number of sets in $\mathcal{F}$ is bounded by $\mathcal{O}(k^{d^2})$. Thus we can design an algorithm similar to that described in Lemma 7.2. □

7.4 Unique Cover

The Unique Cover problem was studied in [Misra 2013] and was found to be FPT. However, the problem does not have a polynomial kernel unless NP $\subseteq$ CoNP/poly, as shown in [Dom 2014]. In this section, we exhibit polynomial sized kernels for several geometric versions, exploiting the geometric property satisfied by each set of the input set system.

Recall that the Unique Cover problem is parameterized by the number of elements that we desire to cover uniquely. To begin with, in the abstract setting, we show that the number of elements in the universe can be bounded by k^2. Note that it is straightforward to bound the sizes of the individual sets in an instance of Unique Cover, with the following observation.

Observation 7.2. *If there exists $F_i \in \mathcal{F}$ such that $|F_i| \geq k$, then the given instance is a* YES *instance, with $\{F_i\}$ serving as a valid solution.*

We now turn to an argument for bounding the size of the universe in an instance of Unique Cover. A tighter bound has been given in [Misra 2013].

Lemma 7.4. Unique Cover *admits a quadratic element kernel.*

Proof. By Observation 7.2, we may assume that every set contains at most $(k-1)$ elements. Let $\mathcal{S} \subseteq \mathcal{F}$ be a maximal subfamily of disjoint sets. Let $U_1 = \bigcup_{S \in \mathcal{S}} S$. Because of disjointness, every set in $\mathcal{S}$ uniquely covers the elements it contains. Therefore, if $|U_1| \geq k$, it is a YES instance.

If $|U_1| < k$, then we proceed as follows. By the maximality of $\mathcal{S}$, $U_1 = \bigcup_{S \in \mathcal{S}} S$ is a hitting set. We delete U_1 from our input instance, reducing the number of elements in the universe by at most $k-1$. The resulting instance does not contain any set from $\mathcal{S}$, and the size of each set in $\mathcal{F} \setminus \mathcal{S}$ is strictly reduced by the removal of U_1.

We repeat the procedure of finding a maximal subfamily of disjoint sets till we either are at a stage where the number of elements covered by the current maximal subfamily is at least k, in which case we correctly output YES, or when every element has been deleted. Since we started with an instance where every set had at most $k-1$ elements, the number of times this procedure is repeated is at most $k-1$. The number of elements in the original input instance is equal to the number of elements removed in the whole algorithm. In each step we delete at most $k-1$ elements.

Thus, either we have resolved the instance during the course of the procedure described above, or the number of elements in the original input instance can be at most $(k-1)^2$, thus we have a polynomial element kernel as desired. □

The following result regarding sets of bounded VC Dimension are now implied by Proposition 7.1 and Lemma 7.4.

Corollary 7.1. Unique Cover *on set systems of VC Dimension bounded by a constant d admits a polynomial kernel.*

Proof. From Lemma 7.4, we know that Unique Cover is FPT with an element kernel of $\mathcal{O}(k^2)$. By Proposition 7.1, the family $\mathcal{F}$ can have at most $\mathcal{O}((k^2)^d)$ sets. Thus, we have a kernel for this version of Unique Cover. □

As an immediate consequence, we get the existence of polynomial kernels in special geometric cases, since these geometric set families have constant VC Dimension.

Corollary 7.2. Unique Cover *admits a polynomial kernel for set systems of lines, hyperplanes, axis-parallel rectangles and disks.*

Proof. The proof follows from the fact that these geometric set families have constant VC Dimension. Following from Proposition 7.1, for a family of sets of VC Dimension d, the number of distinct sets is bounded by $O(n^d)$. In particular, we have the following based on Lemma 7.4.

1. Since the VC Dimension of lines is two, Unique Cover admits a kernel with $O(k^2)$ points and $O(k^4)$ lines.
2. Since the VC Dimension of hyperplanes in $\mathbb{R}^d$ is $d+1$, Unique Cover admits a kernel with $O(k^2)$ points and $O(k^{(2d+1)})$ hyperplanes.
3. Since the VC Dimension of axis-parallel rectangles is four, Unique Cover admits a kernel with $O(k^2)$ points and $O(k^8)$ axis-parallel rectangles.
4. Since the VC Dimension of disks is three, Unique Cover admits a kernel with $O(k^2)$ points and $O(k^6)$ disks.

□

It may be noted that kernels of smaller size are known for lines. Using the fact that two lines intersect in at most one point and Theorem 3 in [Misra 2013], we can see that Unique Cover with lines admits a kernel with $O(k^2)$ lines.

7.5 Unique Set Cover

We show that Unique Set Cover is W[1]-hard. However, as with the other problems, assuming geometric properties on the set family provides positive algorithmic results.

Lemma 7.5. Unique Set Cover *is* W[1]*-hard.*

Proof. We give a reduction from the Multicoloured Clique problem, which is known to be W[1]-hard when parameterized by the size k of the solution clique [Fellows 2009]. An instance of the Multicoloured Clique problem consists of a graph G and a partition of its vertex set into k parts, and the question is if there exists a clique involving exactly one vertex from each part. For an instance (G, k) of Multicoloured Clique, where $V(G) = V_1 \uplus \ldots \uplus V_k$, we construct the following instance for Unique Set Cover:

- We create a set of elements corresponding to the vertices of G, denoted by $A = \{e_v | v \in V(G)\}$. Note that there is a natural partition of $A = A_1 \uplus A_2 \uplus \ldots \uplus A_k$ based on the given partition of $V(G)$.
- We add a set of elements $B = \{b_{ij} | 1 \leq i, j \leq k\}$, and a set of elements $C = \{c_i | 1 \leq i \leq k\}$.
- Let $U = A \cup B \cup C$.
- For each $1 \leq i \leq k$ we add a set $S_i = \{c_i\} \cup \{e_v | v \in V_i\}$ to our set family $\mathcal{F}$.
- For each edge $uv \in E(G)$, where $u \in V_i$ and $v \in V_{j \neq i}$, we add the following set to $\mathcal{F}$:

$$S_{u,v} = \{b_{ij}\} \cup \{e_w | w \in (V_i \setminus u) \cup (V_j \setminus v)\}$$

- We assume, without loss of generality, that $k > 2$ (if not, the problem is solved in constant time and we define an appropriate trivial reduced instance).
- Finally, we set $l = \binom{k}{2} + 2k$. This completes the description of the reduced instance.

In the forward direction, suppose $X = \{v_1, v_2, \ldots, v_k\}$ is a multi-coloured clique in G. The family of sets $\{S_i | 1 \leq i \leq k\} \cup \{S_{v_i,v_j} | v_i, v_j \in X\}$ is a set cover where the l elements in $\{e_{v_i} | v_i \in X\} \cup B \cup C$ are uniquely covered. Thus, this is a solution family for Unique Set Cover.

In the backward direction, since a solution must be a set cover, we must cover c_i for all $1 \leq i \leq k$. Each element of C belongs to a unique set. Hence the subfamily of sets $\{S_1, \ldots, S_k\}$ must be present in any set cover and the k elements of C are uniquely covered by any set cover.

For a fixed $1 \leq i \leq k$, consider the collection of elements $B_i = \{b_{ij} | 1 \leq j \neq i \leq k\}$. Each b_{ij} must be covered by at least one set S_{u^j,v^j}, where $u^j v^j \in E(G)$ and $u^j \in V_i$ while $v^j \in V_j$. Assuming $k > 2$, if each $u^j, 1 \leq j \neq i \leq k$ is the same vertex $u \in V_i$ then the set cover considered so far covers e_u uniquely, since all sets S_{u,v^j} do not contain e_u. On the other hand, suppose $u^p = e_p$ and $u^q = e_q$ for $1 \leq p \neq q \leq k$. Then S_{u^p,v^p} covers all elements of A except for u_p and S_{u^q,v^q} covers all elements of A except for u_q. Since $u_p \neq u_q$, both u_p and u_q are covered at least once by $S_{u^p,v^p} \cup S_{u^q,v^q}$. Recall that these elements are already covered by S_i, and therefore we cannot cover any of the elements of A_i uniquely. This is true for all $1 \leq i \leq k$. Thus, we can only hope to uniquely cover at most k elements from the set A.

The remaining elements are from B, add to $\binom{k}{2}$ and must all be uniquely covered in a solution. Let the set chosen to uniquely cover the element of $b_{ij} \in B$ be $S_{u_{ij},v_{ij}}$. This set corresponds to an edge $u_{ij}v_{ij} \in E(G)$ such that $u_{ij} \in V_i$ and $v_{ij} \in V_j$ for some $1 \leq i \neq j \leq k$.

To meet the given budget, a solution for Unique Set Cover must cover exactly k elements of A uniquely. From the above argument, this can only happen if we cover exactly one element e_{u_i} from each partition A_i. Also, for this to happen, for each i, and each pair $1 \leq j \neq j' \leq k$ such that $j \neq i$ and $j' \neq i$, it must be true that $u_{ij} = u_{ij'} = u_i$.

Now consider the set of vertices $X = \{u_i | 1 \leq i \leq k\} \subseteq V(G)$. This is a set of k vertices where no two vertices are from the same partition of $V(G)$. For any pair $1 \leq i \neq j \leq k$,

the set S_{u_i,u_j} chosen to cover the element b_{ij} ensures that there is an edge $u_iu_j \in E(G)$. Hence, $G[X]$ is a multicoloured clique in G.

This proves the W[1]-hardness of Unique Set Cover. □

The reduction also shows that Unique Set Cover is NP-hard. In fact, even when we consider the special case when the universe U is a set of n points in $\mathbb{R}^2$ and each set is a line, the Unique Set Cover problem turns out to be NP-hard, as we show in our next lemma.

Lemma 7.6. Unique Set Cover *on set systems of lines is* NP*-complete.*

This NP-hardness reduction is very similar to the NP-hardness reduction for Point Line Cover [Megiddo 1982]. Unlike [Megiddo 1982], we reduce the problem from 1-IN-3-SAT, instead of 3-SAT.

Proof. Given a boolean formula F in conjunctive normal form where each clause has three literals, 1-IN-3-SAT asks whether there is a truth assignment to the variables of F such that each clause of F has exactly one true literal. Let ϕ be an instance of 1-IN-3-SAT, with n variables $\{v_1, v_2, \ldots, v_n\}$ and m clauses $\{C_1, \ldots C_m\}$. We construct an instance $(U, \mathcal{F}, k)$ for Unique Set Cover with lines in the following manner:

- For each clause $C_i, 1 \leq i \leq m$, we create a point P_i.
- For each variable $v_j, 1 \leq j \leq n$, we create a grid of m^2 points $p^j_{ab}, 1 \leq a, b \leq m$.
- For a fixed $1 \leq j \leq n$ and a fixed $1 \leq a \leq m$, the set of points $\{p^j_{qa} | 1 \leq q \leq m\}$ are contained in a line L_{ja} and the set of points $\{p^j_{aq} | 1 \leq q \leq m\}$ are contained in a line $\bar{L}_{ja}$.
- For each $1 \leq i \leq m$ and $1 \leq j \leq n$, if the literal v_j is in C_i, then the point P_i is contained in L_{ji}. If the literal $\bar{v}_j$ is in C_i then the point P_i is contained in $\bar{L}_{ji}$.
- The above description gives the exact points that can lie on each line.
- $U = \{P_i | 1 \leq i \leq m\} \cup \bigcup_{1 \leq j \leq n} \{p^j_{ab} | 1 \leq a, b \leq m\}$. $|U| = nm^2 + m$.
- $\mathcal{F} = \{L_{jb} | 1 \leq j \leq n, 1 \leq b \leq m\} \cup \{\bar{L}_{jb} | 1 \leq j \leq n, 1 \leq b \leq m\}$.
- We set $k = nm^2 + m$.

Suppose ϕ is a YES instance for 1-IN-3-SAT and let Γ be a truth assignment. If $\Gamma(v_j) = 1$ we pick the lines $\{L_{jb} | 1 \leq b \leq m\}$. If $\Gamma(v_j) = 0$ we pick the lines $\{\bar{L}_{jb} | 1 \leq b \leq m\}$. By construction, all points $p^j_{ab}, 1 \leq j \leq n, 1 \leq a, b \leq m$, are covered uniquely. Since, Γ is a satisfying assigment for 1-IN-3-SAT, the points $P_i, 1 \leq i \leq m$, are also covered uniquely. Thus, our constructed instance is a YES instance for Unique Set Cover with lines.

Conversely, suppose $(U, \mathcal{F}, k)$ has a solution family $\mathcal{F}'$ for Unique Set Cover. By the value of k, each element in the instance has to be covered uniquely. Each point P_i is contained in exactly 3 lines, by our construction. Suppose the line $L_{ji}, 1 \leq j \leq n$ covers the point P_i uniquely in $\mathcal{F}'$. Then it is only possible to cover the m^2 points

$p^j_{ab}, 1 \leq a, b \leq m$ uniquely by the subfamily $\{L_{jq} | 1 \leq q \leq m\}$. Hence, all these lines must belong to $\mathcal{F}'$. Suppose the line $\bar{L}_{ji}, 1 \leq j \leq n$ covers the point P_i uniquely in $\mathcal{F}'$. Then, by a similar argument, the subfamily $\{\bar{L}_{jq} | 1 \leq q \leq m\}$ must belong to $\mathcal{F}'$. Suppose the line $L_{ji}, 1 \leq j \leq n$ is not picked to cover the point P_i uniquely in $\mathcal{F}'$. Then it is only possible to cover the m^2 points $p^j_{ab}, 1 \leq a, b \leq m$ uniquely by the subfamily $\{\bar{L}_{jq} | 1 \leq q \leq m\}$, and all these lines must belong to $\mathcal{F}'$. Suppose the line $\bar{L}_{ji}, 1 \leq j \leq n$ is not picked to cover the point P_i uniquely in $\mathcal{F}'$. Then, by a similar argument, the subfamily $\{L_{jq} | 1 \leq q \leq m\}$ must belong to $\mathcal{F}'$. From this, we construct a satisfying assignment Γ for ϕ. If the family $\{L_{jb} | 1 \leq b \leq m\}$ is used to cover the points $\{p^j_{ab} | 1 \leq a, b \leq m\}$, then $\Gamma(v_j) = 1$. Otherwise, $\Gamma(v_j) = 0$. Under this assignment, any clause C_i is satisfied only by the literal corresponding to the line uniquely covering P_i. Thus Unique Set Cover with lines is NP-hard. □

This implies that Unique Set Cover on set systems with bounded intersection and Unique Set Cover on set systems of hyperplanes are also NP-hard. In the parameterized context, although the problem is W[1]-hard in the general setting, we look at some special cases where this problem can be solved in FPT time. First, we make a few observations.

Observation 7.3. *Let $\mathcal{F}'$ be a solution for* Unique Set Cover. *Then any minimal set cover contained in $\mathcal{F}'$ is also a solution for* Unique Set Cover.

Thus, it is enough to find a minimal set cover that covers at least k elements uniquely.

Observation 7.4. *Any minimal set cover of size at least k is a* Unique Set Cover *solution for a given instance. In particular, if the minimum set cover of the given instance is of size at least k then it is a* YES *instance for the* Unique Set Cover *problem.*

Now, we turn to algorithmic results for special cases of Unique Set Cover.

Sets of Bounded Intersection. First, we consider set systems $(U, \mathcal{F})$ where $\mathcal{F}$ has the property that for any pair of sets $F_1, F_2 \in \mathcal{F}$, $|F_1 \cap F_2| \leq c$.

Lemma 7.7. Unique Set Cover *for sets of bounded intersection c is* FPT *when parameterized by $c + k$.*

Proof. Construct a minimal set cover $\mathcal{S}$ for the instance. If the size of $\mathcal{S}$ is at least k, then it is a solution for Unique Set Cover, by Observation 7.4. Similarly, if there is a set $S \in \mathcal{S}$ that contains at least k private elements, then too $\mathcal{S}$ is a solution for Unique Set Cover. Suppose there are at most $k-1$ sets in $\mathcal{S}$ and each set has at most $k-1$ private elements. By pigeonhole principle, there must be a set $S \in \mathcal{S}$ which contains at least $n/(k-1)$ elements of U. The number of elements in S that can belong to other sets of $\mathcal{S}$ is at most $c(k-2)$, because of the bounded intersection property. If $\frac{n}{k-1} - c(k-2) \geq k$ then the given instance is a YES instance. Otherwise, $\frac{n}{k-1} - c(k-2) \leq k-1$, which implies that $n \leq (1+c)(k-1)^2$.

Since the sets have bounded pairwise intersection of at most c, any subset of $c+1$ elements can appear together in at most one set. Therefore, the number of sets are bounded by $n^{c+1} \leq ((1+c)(k-1)^2)^{c+1}$. We can now guess the uniquely covered elements, and the distribution of the uniquely covered elements in a Unique Set Cover solution. Finally, we can check whether there are sets in $\mathcal{F}$ to validate the guess in polynomial time. As the number of guesses is an FPT function, the running time of this algorithm is FPT. □

Hyperplanes in $\mathbb{R}^d$. Next, we consider a geometric set system $(U, \mathcal{F})$ where U is a set of n points in $\mathbb{R}^d$ and sets in $\mathcal{F}$ are defined by hyperplanes in $\mathbb{R}^d$. When $d = 2$, these are lines and this is a special case of sets with bounded intersection. For $d > 2$, hyperplanes do not have this property. Nonetheless, we obtain an FPT algorithm for hyperplanes, by reducing the given instance to an instance of Unique Set Cover for sets with bounded intersection.

Lemma 7.8. Unique Set Cover *on set systems of hyperplanes in* $\mathbb{R}^d$ *is* FPT*, where* d *is a fixed constant.*

Proof. Let U be the universe of n elements and $\mathcal{F}$ be the family of m hyperplanes in $\mathbb{R}^d$. The following Reduction Rule aims at reducing the number of points, while maintaining the Unique Set Cover solution if there exists one.

Reduction Rule 7.5. *for* i *from* 1 *to* $d-1$*:*
Suppose $P \subseteq U$ *is a set of at least* k^i *points such that* $dim(P) = i$*, and* P *is contained in at least one hyperplane of* $\mathcal{F}$*. Suppose* $\mathcal{F}_P \subseteq \mathcal{F}$ *is the set of all hyperplanes that contain* P*. If* $\mathcal{F} \setminus \mathcal{F}_P$ *is a set cover for* U*, then we say* YES *for our input instance and exit. Otherwise, we delete all but* k^i *points of* P *from the universe. If a hyperplane becomes empty, we delete that hyperplane from* $\mathcal{F}$*.*

We prove the correctness of this reduction rule by induction on i. When $i = 1$, P is a line with more than k points. We abuse notation and also use P to refer to this collection of points. Suppose $\mathcal{F} \setminus \mathcal{F}_P$ is a set cover for the instance, let $\mathcal{G}$ be a minimal set cover obtained from $\mathcal{F} \setminus \mathcal{F}_P$. Then, by Observation 7.1, any set in $\mathcal{G}$ can contain at most one point of P. To cover all elements of P, there must be at least $k + 1$ sets in $\mathcal{G}$. By Observation 7.4, we correctly say YES. If no such set cover exists, then we know that any set cover for the input instance must contain at least one hyperplane from $\mathcal{F}_P$. Let $P' \subset P$ be the set of all but k points that are deleted by the Reduction Rule and $P'' = P \setminus P'$. Also, let $\mathcal{F}'$ be the family of hyperplanes that became empty and got deleted from $\mathcal{F}$. Suppose $\mathcal{G} = \{H_1, \ldots, H_l\}$ is a minimal solution for $(U, \mathcal{F}, k)$. Since P is a line with more than k points, there must be at least one H_i that contains all of P. Now, the following cases can occur:

1. Suppose there are two planes $H_i, H_j, 1 \leq i \neq j \leq l$, both of which contain the set P. Then none of the points in P are uniquely covered by this solution. The points which are uniquely covered are not deleted as a result of this Reduction Rule. Also, by definition of minimality, no hyperplane of $\mathcal{G}$ could have become empty after this Reduction Rule was applied. Hence, $\mathcal{G}$ remains a solution for Unique Set Cover in $(U \setminus P', \mathcal{F} \setminus \mathcal{F}', k)$.

2. Suppose $l > k$. Since we have dealt with the case when P is covered by at least 2 hyperplanes of $\mathcal{G}$, we can assume that there is exactly one hyperplane in $\mathcal{G}$ that contains P. There are at least k remaining hyperplanes in $\mathcal{G}$. Since, $\mathcal{G}$ is a minimal solution for Unique Set Cover, each of these remaining hyperplanes cover a point uniquely, and none of these uniquely covered points belong to P. Hence, $\mathcal{G}$ remains a solution for $(U \setminus P', \mathcal{F} \setminus \mathcal{F}', k)$.

3. Finally, suppose $l \leq k$, and as before, let $H \in \mathcal{G}$ be the hyperplane that contains all points in P. Then at most $l - 1$ points of P are not uniquely covered. All other points of P must be uniquely covered. In particular, at most $l - 1$ points of $P \setminus P'$

are not uniquely covered , which implies that at least $k-l+1$ points in $P \setminus P'$ are uniquely covered by $\mathcal{G}$. By minimality, for each hyperplane H' in $\mathcal{G} \setminus \{H\}$ there is a point $p_{H'}$ that is uniquely covered by H'. Thus, at least $k-l+1$ points from $P \setminus P'$ and $l-1$ points from $\mathcal{G} \setminus \{H\}$ are covered uniquely. Again, by minimality, no hyperplane of $\mathcal{G}$ could have become empty because of application of the Reduction Rule. Hence, $\mathcal{G}$ is a solution in $(U \setminus P', \mathcal{F} \setminus \mathcal{F}', k)$.

On the other hand, let $\mathcal{G}'$ be a minimal solution for $(U \setminus P', \mathcal{F} \setminus \mathcal{F}', k)$. Assume $\mathcal{G}'$ is not a solution for $(U, \mathcal{F}, k)$. Then each point in P'' is covered by a different set in $\mathcal{G}'$. Let this subfamily of $\mathcal{G}'$, with at least k hyperplanes, be $\mathcal{H}'$.Let $G \in \mathcal{F}_P$. Consider $\mathcal{G}' \cup G$, which is clearly a set cover for $(U, \mathcal{F})$. Let $\mathcal{S} \subset \mathcal{G}' \cup G$ be a minimal set cover of $(U, \mathcal{F})$. Let $P_1 \subseteq (U \setminus P')$ be a set of k points, such that for each hyperplane H in $\mathcal{H}'$ there is a point in P_1 that it uniquely covers with respect to $\mathcal{G}'$. Let $P_2 \subseteq P_1$ be uniquely covered by $\mathcal{S}$. Each point in $P_1 \setminus P_2$ has exactly one hyperplane in $\mathcal{S}$, other than G, containing it. By the minimality of $\mathcal{S}$, this hyperplane has a point that is uniquely covered by it. Therefore, for all of the points in $P_2 \setminus P_1$, either that point is uniquely covered by G or a hyperplane (other than G) in $\mathcal{S}$ containing it is uniquely covering another point. Therefore, $\mathcal{S}$ still covers at least k points uniquely and is a solution for $(U, \mathcal{F}, k)$.

Now, assume $i > 1$ and the Induction Hypothesis is true for all $j < i$. Suppose P is a set of at least k^i points such that $dim(P) = i$, and P is contained in at least 1 hyperplane. Suppose there is a set cover $\mathcal{G}$ for the instance, contained in $\mathcal{F} \setminus \mathcal{F}_P$. Then any set in $\mathcal{G}$ can intersect P at a subset $Q \subseteq P$ such that $dim(Q) < i$ (Observation 7.1). By Induction Hypothesis, $|Q| \leq k^{i-1}$. Thus, $\mathcal{G}$ must have at least $k+1$ hyperplanes to cover all points of P. By Observation 7.4, we correctly output YES. Otherwise, every set cover of the input instance must contain at least one hyperplane from $\mathcal{F}_P$. Let $P' \subset P$ be the set of all but k^i points that are deleted by the Reduction Rule. Also, let $\mathcal{F}'$ be the family of hyperplanes that became empty and got deleted from $\mathcal{F}$. Suppose $\mathcal{G} = \{H_1, \ldots, H_l\}$ be a minimal solution for $(U, \mathcal{F}, k)$. The following cases can occur:

1. Suppose there are 2 planes $H_i, H_j, 1 \leq i \neq j \leq l$, both of which contain the set P. Then none of the points in P are uniquely covered by this solution. The points which are uniquely covered cannot be deleted as a result of this Reduction Rule. Also, by definition of minimality, no hyperplane of $\mathcal{G}$ could have become empty after this Reduction Rule was applied. Hence, $\mathcal{G}$ remains a solution for Unique Set Cover in $U \setminus P', \mathcal{F} \setminus \mathcal{F}', k)$.

2. Suppose $l > k$. Since we have dealt with the case when P is covered by at least 2 hyperplanes of $\mathcal{G}$, we can assume that there is exactly one hyperplane in $\mathcal{G}$ that contains P. There are at least k remaining hyperplanes in $\mathcal{G}$. Since, $\mathcal{G}$ is a minimal solution for Unique Set Cover, each of these remaining hyperplanes cover a point uniquely, and none of these uniquely covered points belong to P. Hence, $\mathcal{G}$ remains a solution for $U \setminus P', \mathcal{F} \setminus \mathcal{F}', k)$.

3. Lastly, suppose P is covered by at most one hyperplane, say H_1 of $\mathcal{G}$, and $l \leq k$. For each $2 \leq i \leq l$, the intersection set $Q = H_i \cap P$ can have at most k^{i-1} points, by Observation 7.1 and induction hypothesis. Then at most $(l-1) \cdot k^{i-1}$ points of P are not uniquely covered. All other points of P must be uniquely covered. In particular, at most $(l-1) \cdot k^{i-1}$ points of $P \setminus P'$ are not uniquely covered , which implies that at least $k-l+1$ points in $P \setminus P'$ are uniquely covered by $\mathcal{G}$. By

minimality, for each hyperplane H' in $\mathcal{G} \setminus \{H_1\}$ there is a point $p_{H'}$ that is uniquely covered by H'. Thus, at least $k - l + 1$ points from $P \setminus P'$ and $l - 1$ points from $\mathcal{G} \setminus \{H\}$ are covered uniquely. Again, by minimality, no hyperplane of $\mathcal{G}$ could have become empty because of application of the Reduction Rule. Hence, $\mathcal{G}$ is a solution in $U \setminus P', \mathcal{F} \setminus \mathcal{F}', k)$.

On the other hand, let $\mathcal{G}'$ be a minimal solution for $U \setminus P', \mathcal{F} \setminus \mathcal{F}', k)$. Assume $\mathcal{G}'$ is not a solution for $(U, \mathcal{F}, k)$. By Induction Hypothesis, all the k^i points in P'' are covered by a subfamily of at least k different hyperplanes, say $\mathcal{H}'$, in $\mathcal{G}'$. Let $P_1 \subseteq (U \setminus P')$ be a set of k points, such that for each hyperplane H in $\mathcal{H}'$ there is a point in P_1 that it uniquely covers with respect to $\mathcal{G}'$. Let $G \in \mathcal{F}_P$. Let $\bar{\mathcal{F}} \subset \mathcal{G}' \cup G$ be a minimal set cover of $(U, \mathcal{F})$. Let $P_2 \subseteq P_1$ be the set of points that are uniquely covered by $\bar{\mathcal{F}}$. By an argument similar to the base case, for every point in $P_1 \setminus P_2$, either it is uniquely covered by G or it is contained in a hyperplane in $\bar{\mathcal{F}}$ that uniquely covering at least one other point. Therefore $\bar{\mathcal{F}}$ still covers at least k points uniquely and is a solution for $(U, \mathcal{F}, k)$.

We exhaustively apply this Reduction Rule. At the end, any hyperplane contains at most k^{d-1} points. Let $\mathcal{G}$ be a minimal set cover for the instance. If there are at least $k + 1$ hyperplanes in $\mathcal{G}$, then due to Observation 7.4, we correctly say YES. Otherwise, there are at most k hyperplanes in the set cover $\mathcal{G}$, which implies that $|U| \leq k^{(d-1)} \cdot k$. The number of hyperplanes that can contain these points is at most $(k^d)^d$. Thus, we have a kernel for the problem. For the algorithm, we guess k points $P \subseteq U$ that are uniquely covered by a solution and the family $\mathcal{G}$ of at most k hyperplanes that are responsible for this unique coverage. Let $\mathcal{F}^P = \{H | H \in \mathcal{F} \setminus \mathcal{G}, \exists p \in P \text{ s.t } p \in H\}$. We check whether the family $\mathcal{F} \setminus \mathcal{F}^P$ is a set cover or not. There are at most $\mathcal{O}(k^{kd^2})$ possible pairs $(P, \mathcal{G})$. Thus the problem is FPT. □

7.6 Chapter Summary

In this chapter, we looked at several variants of Set Cover, and considered the problems in different geometric settings. For the geometric settings that we considered, we were able to settle the question of which class in the W-hierarchy the problem belongs to. For most of the problems that are in FPT, we could provide polynomial kernels. It would be interesting to ask for the parameterized complexity of Exact Cover for disks, or even unit disks. The parameterized complexity of Unique Set Cover for a set system of unit disks, or of unit squares is also open.

Chapter 8

Exact and FPT Algorithms for Max-Conflict Free Colouring in Hypergraphs

8.1 Introduction

As mentioned earlier, a hypergraph is another name for a set system. A hypergraph H is denoted by a pair $(U, \mathcal{F})$, where U is a set of n vertices and $\mathcal{F}$ contains m subsets of U. We call these subsets *hyperedges*. Thus a general graph is a hypergraph where every hyperedge contains exactly two vertices. A k-vertex-colouring of H, for $k \in \mathbb{N}$ is a function $c : U \to \{1, 2, \ldots, k\}$. A colouring is called a *proper* colouring if none of the hyperedges are monochromatic, i.e. all the vertices of the hyperedge are not of the same colour. We look at a stricter version of colouring called conflict-free colouring.

Definition 8.1. *A vertex colouring $c : U \to \{1, 2, \ldots, k\}$ of a hypergraph $H = (U, \mathcal{F})$ is said to be* conflict-free*, if for every $F \in \mathcal{F}, \exists v \in F$ such that $\forall u \in F$, $u \neq v$ implies $c(u) \neq c(v)$. In other words, every hyperedge has a uniquely coloured vertex.*

The minimum number of colours required to conflict-free colour the vertices of a hypergraph H is called the *conflict-free chromatic number* of H and is represented as $\chi_{cf}(H)$. For a given hypergraph H, the *minimum conflict-free colouring problem* refers to computing the value of $\chi_{cf}(H)$.

The concept of conflict-free colouring was introduced for hypergraphs induced by geometric regions, motivated by the frequency allocation problem in cellular networks [Even 2002]. This problem also found applications in areas like Radio Frequency Identification and Robotics. Conflict-free colouring question has been extensively studied for hypergraphs induced by various geometric regions [Ajwani 2012, Har-Peled 2005, Smorodinsky 2007].

Pach and Tardos [Pach 2009] initiated the study of conflict-free colouring for general hypergraphs and gave an upper bound of $O(\sqrt{m})$ on the conflict-free chromatic number. On the algorithmic side, the minimum conflict-free colouring problem for a general hypergraph is NP-hard by results shown in [Even 2002, Gargano 2015]. [Pach 2009] also studied the conflict-free colouring of hypergraphs induced by graph neighbourhoods. Here the vertex set of the hypergraph corresponds to vertex set of a general graph $G = (V, E)$ and the hyperedges are defined by the neighbourhoods (open or closed) of the vertices in G. [Pach 2009] showed an upperbound of $O(\log^2 n)$ and a lower bound of $\Omega(\log n)$ for this problem. Gargano and Rescigno [Gargano 2015] studied the minimum conflict-free colouring of hypergraphs induced by graph neighbourhoods (both open and closed) and

showed NP- completeness. [Gargano 2015] also showed that the minimum conflict-free colouring problem for these graphs becomes tractable when parameterized by the vertex cover or the neighbourhood diversity number of the graph. Specifically, they gave an algorithm that decides whether a hypergraph induced by neighbourhoods of a graph G can be conflict-free coloured using k colours. This algorithm runs in time $2^{\mathcal{O}(kt \log k)}$ where t represents the neighbourhood diversity number of G. Note that this also implies an algorithm to solve the minimum conflict-free colouring problem in hypergraphs induced by graph neighbourhoods, which runs in $\mathcal{O}(n^n)$ time.

In [Ashok 2015a], we initiate a study of a maximization version of the MINIMUM CONFLICT-FREE COLOURING problem.

MAXIMUM CONFLICT-FREE COLOURING(MAX-CFC)
Input: A hypergraph $(U, \mathcal{F})$ on n vertices and m hyperedges, and an integer $r \geq 2$.
Output: A maximum-sized subfamily of hyperedges that can be conflict-free coloured with r colours.

The NP-hardness of this problem follows from the NP-hardness reductions shown in [Gargano 2015]. We give an exact algorithm for this problem that runs in $\mathcal{O}(2^{m+n}) \cdot n^{\mathcal{O}(1)}$ time. As a corollary, we obtain an exact algorithm, of running time $\mathcal{O}(4^n) \cdot n^{\mathcal{O}(1)}$, for hypergraphs induced by neighbourhoods in graphs. We also define a stronger variant of conflict-free colouring namely, *unique-maximum colouring* [Cheilaris 2011].

Definition 8.2. *A vertex colouring $c : U \to \{1, 2, \ldots, k\}$ is said to be* unique-maximum, *if for every $F \in \mathcal{F}, \exists v \in F$ such that $\forall u \in F$, $u \neq v$ implies $c(u) < c(v)$. In other words, the maximum colour occurring in a hyperedge occurs uniquely. The minimum number of colours required to unique-maximum colour H is called the unique-maximum chromatic number of H.*

A vertex of a hyperedge $h \in \mathcal{F}$ is said to be unique-maximum coloured if it is the unique vertex that is coloured with the maximum colour occurring in the the hyperedge h. For a given hypergraph H, the *minimum unique-maximum colouring problem* refers to computing the minimum number of colours required to unique-maximum colour H.

Similar to the definition of MAX-CFC, we can define MAXIMUM UNIQUE-MAXIMUM COLOURING (MAX-UMC) to take as input a hypergraph H and a positive integer $r \geq 2$, and output the largest subfamily of hyperedges that has a unique-maximum colouring with r colours. Our algorithms for MAX-CFC, with some modification, also works for MAX-UMC.

In the parameterized setting, we study MAX-CFC parameterized by the solution size.

P-CFC **Parameter:** k
Input: A hypergraph $(U, \mathcal{F})$ on n vertices and m hyperedges, and positive integers $r \geq 2$ and k.
Question: Is there a subfamily of at least k hyperedges that can be conflict-free coloured using r colours?

We also study this problem when we restrict the input hypergraph to that induced by the closed/open neighbourhood of a graph G. Similarly, P-UMC is defined and studied.

Our Results and Methods. The results of this chapter are from [Ashok 2015a]. In the

realm of parameterized algorithms, we obtain the following result.

1. We show that the problem is FPT by designing a kernel with at most 4^k vertices and $\mathcal{O}(k \log k)$ hyperedges. The kernel is obtained by finding a novel connection to Unique Coverage problem [Misra 2013]. We use this one way connection to either say that the given instance for p-CFC is a YES instance or conclude that the number of hyperedges is upper bounded by $\mathcal{O}(k \log k)$. Finally, using extremal results on set-family we bound the number of vertices (elements) to 4^k. Moreover, when we restrict the input hypergraph to that induced by the closed/open neighbourhood of a graph G, then the above imply polynomial kernels for these variants.
2. A direct consequence of our kernel is an $r^{4^k}(n+m)^{\mathcal{O}(1)}$ algorithm for p-CFC. We exploit the fact that the number of hyperedges is at most $\mathcal{O}(k \log k)$ in the reduced instance to design an FPT algorithm with running time $2^{\mathcal{O}(k \log\log k + k \log r)}(n+m)^{\mathcal{O}(1)}$. We arrive at the required algorithm by combining the fact that we have small number of hyperedges and using the technique of colour coding introduced in [Alon 1995] in a non-trivial manner.
3. All the above results, with minor modifications, hold for p-UMC.

Finally, we design an exact algorithm that solves the Max-CFC problem for general hypergraphs. This algorithm exploits structural properties of a YES instance for Max-CFC. Our algorithm runs in $\mathcal{O}(2^{m+n})$ time. The algorithm also works for the Minimum Conflict-Free colouring problem. In particular, for hypergraphs induced by graph neighbourhoods, our algorithm runs in time $\mathcal{O}(4^n)$ which is a non-trivial improvement over the best known exact algorithm that runs in $\mathcal{O}(n^n)$ time [Gargano 2015]. The algorithm is based on dynamic programming combined with an application of subset-convolution. We refer to [Fomin 2010] for a more detailed introduction to exact algorithms. Some minor modifications to our algorithm give an exact algorithm for Unique Maximum Colouring.

8.2 Preliminaries

Fast Subset Convolution Computation. Suppose we are given a universe U with n elements. The subset convolution of two functions $f, g : 2^U \to \mathbb{Z}$ is a function $(f * g) : 2^U \to \mathbb{Z}$ such that for every $Y \subseteq U$, $(f * g)(Y) = \Sigma_{X \subseteq Y} f(X)g(Y - X)$. It is equivalent to saying that $(f * g)(Y) = \Sigma_{A \uplus B = Y} f(A)g(B)$.

Proposition 8.1. . *For two functions $f, g : 2^U \to \mathbb{Z}$, given all the 2^n values of f and g in the input, all the 2^n values of the subset convolution $f * g$ can be computed in $\mathcal{O}(2^n \cdot n^3)$ arithmetic operations.*

In fact, the result can be extended to subset convolution of functions that map to any ring, instead of $(\mathbb{Z}, +, \times)$. Consider the set $\mathbb{Z} \cup \{\infty\}$, with the added relation that $\forall z \in \mathbb{Z}, \{\infty\} > z$. The *min* operator takes two elements from this set and outputs the minimum of the two elements. Notice that $\mathbb{Z} \cup \{\infty\}$, along with *min* as an additive operator and $+$ as a multiplicative operator, forms a semi-ring. We will call this semi-ring the integer min-sum semi-ring. The subset convolution of two functions $f, g : 2^U \to \mathbb{Z} \cup \{\infty\}$, with *min* and $+$ as the additive and multiplicative operators, becomes $(f * g)(Y) = min_{A \uplus B = Y} f(A) + g(B)$.

Proposition 8.2. *Given two functions $f, g : 2^U \to \{-M, \ldots, M\}$, all the 2^n values of f and g in the input, and all the 2^n values of the subset convolution $(f * g)$ over the integer min-sum semiring can be computed in time $2^n n^{\mathcal{O}(1)} \cdot \mathcal{O}(M \log M \log\log M)$.*

For more details about subset convolutions and fast calculations of subset convolutions, please refer to [Fomin 2010].

8.3 FPT Algorithm for p-CFC

We are given a hypergraph $\mathcal{H} = (U, \mathcal{F})$ as input and two positive integers, k and r. In this section, we give an FPT algorithm for p-CFC on hypergraphs, parameterized by k. In other words, we wish to find out if k hyperedges can be conflict-free coloured using r colours. *For simplicity, throughout this section, we assume that we are given a simple hypergraph, that is no hyperedges are repeated.* We first give a kernel and then use this kernel to get the desired FPT algorithm.

Kernel for p-CFC. We begin with a simple observation that if $r > k$, then we can conflict-free colour any subfamily of k edges with r colours. Thus, for the remaining section, we assume that $r \leq k$.

We can also preprocess the input instance to detect simple YES instances of the problem, by applying the following reductions to the instance.

Reduction Rule 8.1. *If there is a vertex $v \in U$ such that $\mathsf{deg}_H(v)$ is at least k, say* YES.

Lemma 8.1. *Reduction Rule 8.1 is safe.*

Proof. Reduction Rule 8.1 is safe since we can obtain a 2 conflict-free colouring for at least k hyperedges in the following way: assigning the first colour to v and the second colour to all the other vertices gives us at least k conflict-free coloured hyperedges. □

Next, we draw a connection between p-CFC and the Unique Hitting Set (UHS) problem. In UHS, we take a hypergraph H and a positive integer k as input. The question is to decide whether there is a set S of vertices and a subfamily $\mathcal{F}'$ of at least k hyperedges such that each hyperedge in $\mathcal{F}'$ contains exactly 1 vertex from S. In other words, each hyperedge of $\mathcal{F}'$ needs to be uniquely hit by S.

Observation 8.1. *Given a hypergraph H and a positive integer k, if (H, k) is a* YES *instance for* UHS*, then $(H, k, r = 2)$ is a* YES *instance for* p-CFC*.*

Proof. Let S be a solution for (H, k) as an instance for UHS, and let $\mathcal{F}'$ be a set of at least k hyperedges that are uniquely hit by S. We colour the vertices of S with the first colour and the vertices of $U \setminus S$ with the second colour. This colouring function conflict-free colours all hyperedges of $\mathcal{F}'$. Thus, $(H, k, 2)$ is a YES instance for p-CFC. □

The UHS problem, in turn, is related to the Unique Coverage (UC) problem. In UC, we take a hypergraph H and a positive integer k as input. The question is to decide

whether there is a subfamily $\mathcal{F}'$ of hyperedges and a set S of at least k vertices such that each vertex in S belongs to exactly 1 hyperedge of $\mathcal{F}'$. In other words, each vertex of S needs to be uniquely covered by $\mathcal{F}'$.

Lemma 8.2. *An instance* $(H = (U, \mathcal{F}), k)$ *of* UHS *has an equivalent instance* $(H' = (\hat{U}, \hat{\mathcal{F}}), k)$ *of* UC*, where the parameter remains the same, and* $|U| = |\hat{\mathcal{F}}|, |\hat{U}| = |\mathcal{F}|$.

Proof. Given the instance (H, k) of UHS, we construct the equivalent instance $(H' = (\hat{U}, \hat{\mathcal{F}}), k)$ of UC in the following manner:

- For every hyperedge $h \in \mathcal{F}$, we create a new vertex u_h. $\hat{U} = \{u_h \mid h \in \mathcal{F}\}$. $|\hat{U}| = |\mathcal{F}|$.
- For each vertex $v \in U$, let $\mathcal{F}_v = \{h \mid h \in \mathcal{F}, v \in h\}$. Define $T_v = \{u_h \mid h \in \mathcal{F}_v\}$. $\hat{\mathcal{F}} = \{T_v \mid v \in U\}$. Thus each hyperedge in H' corresponds to a vertex of H and $|U| = |\hat{\mathcal{F}}|$

Suppose S was a solution of UHS for (H, k) and let $\mathcal{F}'$ be the set of at least k hyperedges that are uniquely hit by S. Then, consider the subset $\hat{U}_{\mathcal{F}'} \subseteq \hat{U}$, where $\hat{U}_{\mathcal{F}'} = \{u_h \mid h \in \mathcal{F}'\}$, and the subfamily $\hat{\mathcal{F}}_S \subseteq \hat{\mathcal{F}}$, where $\hat{\mathcal{F}}_S = \{T_v \mid v \in S\}$. By the property of S and $\mathcal{F}'$, every vertex of $\hat{U}_{\mathcal{F}'}$, which is of size at least k, is contained in exactly one hyperedge of $\hat{\mathcal{F}}_S$ and therefore (H', k) is a YES instance of UC.

Similarly, let $\hat{\mathcal{F}'}$ be a solution of UC for (H', k) and let $\hat{S}$ be the set of at least k vertices that are uniquely covered by $\hat{\mathcal{F}'}$. Then, consider the subfamily $\mathcal{F}_{\hat{S}} \subseteq \mathcal{F}$, where $\mathcal{F}_{\hat{S}} = \{h \mid u_h \in \hat{S}\}$, and the subset $U_{\hat{\mathcal{F}'}} \subseteq U$, where $U_{\hat{\mathcal{F}'}} = \{v \mid T_v \in \hat{\mathcal{F}'}\}$. By the property of $\hat{S}$ and $\hat{\mathcal{F}'}$, the vertex set $U_{\hat{\mathcal{F}'}}$ uniquely hits $\mathcal{F}_{\hat{S}}$, which is of size at least k. Therefore (H, k) is a YES instance of UHS. □

The UC problem has been studied in the field of parameterized complexity. When k, the number of vertices to be uniquely covered, is the parameter, the problem was shown to be in FPT in [Misra 2013]. The following Proposition was proved in [Misra 2013], and we will shortly show how this is useful to us.

Proposition 8.3. [Misra 2013, Lemma 17] *Let* $(H = (U, \mathcal{F}), k)$ *be an instance of* UC *such that every hyperedge has size at most* $k-1$*. Then there exists a constant* α_{uc} *such that if* $|U| \geq \alpha_{uc} k \log k$ *then* $(H = (U, \mathcal{F}), k)$ *is a* YES *instance and furthermore in polynomial time, it is possible to find a subfamily covering at least* k *elements uniquely.*

We use Proposition 8.3 to bound the universe size for p-CFC.

Lemma 8.3. *Let* $(H = (U, \mathcal{F}), k, r)$ *be an instance of* p-CFC*. Then in polynomial time, either we can conclude that* (H, k, r) *is a* YES *instance of* p-CFC *or* $|\mathcal{F}| \leq \alpha_{uc} k \log k$.

Proof. Let $(H = (U, \mathcal{F}), k, r)$ be an instance of p-CFC. We first check whether Reduction 8.1 applies. If it does not apply then we know each element of U appears in at most $k-1$ sets. Now we consider (H, k) as an instance for UHC and apply the reduction given in Lemma 8.2 to obtain an equivalent instance $(H' = (\hat{U}, \hat{\mathcal{F}}), k)$ of UC. Observe that since each element of U appears in at most $k-1$ sets, we have that every hyperedge in $\hat{\mathcal{F}}$ has size $k-1$. Furthermore, since H is a simple hypergraph no elements in $\hat{U}$ repeat. Now we

apply Proposition 8.3 on $(H' = (\hat{U}, \hat{\mathcal{F}}), k)$. This tells us that either $|\hat{U}| = |\mathcal{F}| \leq \alpha_{uc} k \log k$ or $(H' = (\hat{U}, \hat{\mathcal{F}}), k)$ is a YES instance of UC. In the latter case, combining Lemma 8.2 and Observation 8.1 we have that $(H = (U, \mathcal{F}), k, r)$ is a YES instance of p-CFC. □

Thus, from now onwards, we assume our instance to have at most $\mathcal{O}(k \log k)$ hyperedges. Using an extremal result on set systems [Jukna 2011], we obtain the following.

Theorem 8.1. p-CFC *has a kernel with at most* 4^k *vertices and* $\mathcal{O}(k \log k)$ *sets.*

For the proof of Theorem 8.1, we use the following result. The following definition is required for our purpose. Given a family of sets $\mathcal{F} = \{F_1, ..., F_m\}$, an m-tuple $(x_1, ..., x_m)$ is said to be a strong system of distinct representatives if all the elements x_i are distinct and $x_i \in F_i$ for all $i = 1, 2, ..., m.$ and $x_i \notin F_j$ for all $i \neq j$. We use the following result from [Jukna 2011, Theorem 8.12].

Proposition 8.4. *In any family of more than* $\binom{r+s}{s}$ *sets of cardinality at most* r*, at least* $s + 2$ *of its members have a strong system of distinct representatives.*

Proof of Theorem 8.1. Given a hypergraph $H = (U, \mathcal{F})$, we consider a family of sets $\mathcal{F}'$: For each vertex in $u \in U$, we define the set $\mathcal{F}_u = \{F \in \mathcal{F} \mid u \in F\}$ in $\mathcal{F}'$. Clearly, $\mathcal{F}'$ has n sets, one for each vertex. Since the degree of every vertex $u \in U$ is bounded by k, the size of F_u is also bounded by k. Suppose there exists a strong system of distinct representatives for k members of $\mathcal{F}'$ i.e, there exists k hyperedges in $\mathcal{F}$ such that each of them has a vertex that does not appear in any other hyperedge. Then, by colouring these vertices by colour 1 and giving everything else a different colour, we can 2 conflict-free colour these k hyperedges. Now by substituting $r = k - 1$ and $s = k - 2$ in the statement of Proposition 8.4, we know that if $\mathcal{F}'$ has more than $\binom{2k}{k} \leq 2^{2k}$ sets i.e., if U has more than 2^{2k} vertices, we can say YES. Thus, combining this and Lemma 8.3, we get our result. □

We also get the following corollary.

Corollary 8.1. p-CFC *for hypergraphs induced by graph neighbourhoods admits polynomial kernels.*

Corollary 8.1 follows from Lemma 8.3 and the fact that the number of hyperedges are same as the number of vertices in hypergraphs induced by graph neighbourhoods.

Theorem 8.1 immediately implies that p-CFC is FPT. Given an instance $(H = (U, \mathcal{F}), k, r)$ of p-CFC, by using Theorem 8.1, we either conclude that $(H = (U, \mathcal{F}), k, r)$ is a YES instance of p-CFC or we have that $|U| \leq 4^k$. Now we look at every r-partition of U and check whether there are k hyperedges that are conflict-free coloured. If we succeed for any partition then we return YES, else we conclude that the given instance is a NO instance. The running time of this algorithm is upper bounded by $r^{4^k}(|U| + |\mathcal{F}|)^{\mathcal{O}(1)}$.

Faster FPT algorithm for p-CFC. Let $N = |U| + |\mathcal{F}|$. In this section, we give the full description of an FPT algorithm for p-CFC that runs in $2^{\mathcal{O}(k \log \log k + k \log r)} \cdot N^{\mathcal{O}(1)}$ time. We will assume that our input instance contains at most $\mathcal{O}(k \log k)$ hyperedges and 4^k vertices.

We first define some related concepts. Given a set $S \subseteq U$, a subfamily $\mathcal{F}'$, and a colouring $\Gamma : U \to [r]$, we say that S is a *cfc-solution* if each hyperedge h in $\mathcal{F}'$ is conflict-free coloured and a uniquely coloured vertex of h belongs to S. Furthermore, given such a set S and a hyperedge h, let $\mathsf{unicolelt}_S(h)$ denote the uniquely coloured vertex of h that belongs to S. In what follows, we define an auxiliary problem and give an FPT algorithm for this problem. Finally, we reduce our problem to this one with some guesses and by using the colour coding technique, introduced by Alon et. al. in [Alon 1995], to obtain the desired algorithm for p-CFC.

Partitioned p-CFC	**Parameter:** $r + p + \lvert\mathcal{F}\rvert$
Input: A hypergraph $(U = U_1 \uplus U_2 \cdots U_p, \mathcal{F})$, a function $\Psi_{\mathsf{family}} : \mathcal{F} \to [r]$, $\Psi_{\mathsf{parts}} : [p] \to [r]$, a subset $U' \subseteq U$ and a colouring function $\Gamma' : U' \to [r]$, for every $v \in U \setminus U'$, a list $L_v \subseteq [r]$	
Question: Does there exist a colouring function $\Gamma : U \to [r]$ such that: Each hyperedge is conflict-free coloured, $\Gamma(U') = \Gamma'(U')$. For each $v \in U \setminus U', \Gamma(v) \in L_v$. Also, there exists a cfc-solution set S of size exactly p, for all $i \in [p]$, $\lvert S \cap U_i\rvert = 1$ and for every $h \in \mathcal{F}$, $\mathsf{unicolelt}_S(h) \in \bigcup_{j \in \Psi^{-1}_{\mathsf{parts}}(\Psi_{\mathsf{family}}(h))} U_j$?	

In simple words, the problem definition can be explained as follows. We are given a partitioning of the universe U into p-parts and a partial colouring function Γ' on a subset U'. We are looking for a colouring $\Gamma : U \to [r]$ which extends Γ'. Each vertex v in $U \setminus U'$ has a list of admissible colours, and Γ must choose a colour from L_v. Also, due to Γ, each hyperedge is conflict-free coloured and there exists a cfc-solution set S such that it contains exactly one vertex from each part. Suppose the hypothetical set S is $\{x_1, x_2, \ldots, x_p\}$ (think of x_i as a variable) where $x_i \in U_i$. The function Ψ_{parts} is used to guess the colour of x_i in Γ. The function Ψ_{family} divides the family $\mathcal{F}$ into r *chunks* (not to be confused with parts and colouring). The idea is that the uniquely coloured vertex of $h \in \mathcal{F}$, say x_j, has been assigned the same colour by Γ as h has been assigned to the chunk number by Ψ_{family}, i.e, $\Gamma(x_j) = \Psi_{\mathsf{family}}(h)$. Next we show how we can solve the Partitioned p-CFC problem.

Given an instance $((U = U_1 \uplus U_2 \cdots U_p, \mathcal{F}'), \Psi_{\mathsf{family}}, \Psi_{\mathsf{parts}}, U', \Gamma', \{L_v \subseteq [r] | v \in U \setminus U'\})$ of Partitioned p-CFC, we first do a polynomial time preprocessing of the instance. For all $v \in U'$, we must set $\Gamma(v) = \Gamma'(v)$. In the following Reduction Rules, we show that the input functions Ψ_{family} and Ψ_{parts} allow us to prune the list of some of the vertices. The first reduction rule deals with hyperedges h where $\lvert\Gamma'^{-1}(\Psi_{\mathsf{family}}(h)) \cap h\rvert = 1$

Reduction Rule 8.2. *Suppose there is a hyperedge h containing a unique $w \in U'$ such that $\Psi_{\mathsf{family}}(h) = \Gamma'(w)$. Then, for every $v \in h \setminus \{w\}$ we delete $\Psi_{\mathsf{family}}(h)$ from L_v. We delete h from $\mathcal{F}$.*

Lemma 8.4. *Reduction Rule 8.2 is safe.*

Proof. Suppose S is a potential cfc-solution for the given instance. Let $\mathsf{unicolelt}_S(h) = x$. This means that no other vertex of h should be assigned the colour $\Gamma(x)$. Since $x \in \bigcup_{j \in \Psi^{-1}_{\mathsf{parts}}(\Psi_{\mathsf{family}}(h))} U_j$, it implies $\Gamma(x) = \Psi_{\mathsf{family}}(h)$. As $\Psi_{\mathsf{family}}(h) = \Gamma'(w)$, any satisfying assignment Γ must have $\Gamma(x) = \Gamma'(w)$, and it must be the case that $x = w$. Then, all other vertices $v \in U \setminus U'$ of h must get a colour different from $\Gamma(w) = \Psi_{\mathsf{family}}(h)$. Thus, we have identified the vertex that will determine conflict-free colouring of h and ensured

that no other vertex of h can destroy the uniqueness of w. Thus, in the reduced instance, we can delete h and for every $v \in h \setminus \{w\}$, we remove $\Psi_{\mathsf{family}}(h)$ from L_v.

On the other hand, suppose the reduced instance has a satisfying colouring Γ. For each $v \in U \setminus U'$, the list of admissible colours of the original instance is a superset of the list of admissible colours from the reduced instance. Since $w \in U'$ in the reduced instance, $\Gamma(w) = \Gamma'(w)$. But in the original instance, the function Γ' was the same as in the reduced instance. Therefore, $\Psi_{\mathsf{family}}(h)$ of the original instance is same as $\Gamma(w)$ of the reduced instance. Also, in the reduced instance, no other vertex that belonged to h contains the colour $\Gamma(w)$ in its list. Hence, in the original instance, the same assignment Γ will conflict-free colour h as well, and is a satisfying assignment of the original instance. Thus, Reduction Rule 8.2 is safe. □

We can further reduce the size of the lists by the following Reduction.

Reduction Rule 8.3. *If there is a vertex $v \in U_i, i \in [p]$, and $h \in \mathcal{F}$, such that $v \in h$, $\Psi_{\mathsf{family}}(h) \neq \Psi_{\mathsf{parts}}(i)$, then we remove the colour $\Psi_{\mathsf{family}}(h)$ from the list of v.*

Lemma 8.5. *Reduction Rule 8.3 is safe.*

Proof. Suppose, in a potential cfc-solution S, the uniquely coloured vertex of h was x_j. By definition, $x_j \in \bigcup_{i \in \Psi^{-1}_{\mathsf{parts}}(\Psi_{\mathsf{family}}(h))} U_i$. In other words, x_j belongs to a part U_i such that $\Psi_{\mathsf{parts}}(i) = \Psi_{\mathsf{family}}(h)$ and $\Gamma(x_j) = \Psi_{\mathsf{family}}(h)$. However, for the given vertex v and hyperedge h, $\Psi_{\mathsf{family}}(h) \neq \Psi_{\mathsf{parts}}(i)$ and hence, $v \notin \bigcup_{i \in \Psi^{-1}_{\mathsf{parts}}(\Psi_{\mathsf{family}}(h))} U_i$. Thus, $v \neq x_j$. Since x_j is uniquely coloured in h, $\Gamma(v) \neq \Gamma(x_j)$. Therefore, $\Gamma(v) \neq \Psi_{\mathsf{family}}(h)$ and we can safely delete $\Psi_{\mathsf{family}}(h)$ from L_v in the reduced instance. On the other hand, the reduced instance has admissible colour lists which are subsets of the admissible colour lists of the original instance. Suppose the reduced instance had a colouring Γ such that each hyperedge is conflict-free coloured, and there exists a cfc-solution set S of size exactly p, for all $i \in [p]$, $|S \cap U_i| = 1$ and for every $h \in \mathcal{F}$, $\mathsf{unicolelt}_S(h) \in \bigcup_{j \in \Psi^{-1}_{\mathsf{parts}}(\Psi_{\mathsf{family}}(h))} U_j$. This Γ is satisfying colouring for the original instance as well. Thus, Reduction Rule 8.3 is safe. □

The next rule deals with hyperedges h where $|\Gamma'^{-1}(\Psi_{\mathsf{family}}(h)) \cap h| \geq 2$.

Reduction Rule 8.4. *If there are two vertices $v, w \in U'$ and a hyperedge $h \in \mathcal{F}$, such that $\Psi_{\mathsf{family}}(h) = \Gamma'(v) = \Gamma'(w)$, then we say* NO.

Lemma 8.6. *Reduction Rule 8.4 is safe.*

Proof. Suppose, in a potential cfc-solution S, the uniquely coloured vertex of h was x. By definition, $\Gamma(x) = \Psi_{\mathsf{family}}(h)$. Also, by uniqueness of x, no other vertex of h should be assigned the colour $\Gamma(x)$. However, in our instance, there are already two vertices v, w of h which have been assigned $\Psi_{\mathsf{family}}(h)$ by Γ'. This means that there h cannot be conflict-free coloured by any satisfying assignment Γ. Thus, we correctly say NO. □

Reduction Rule 8.5. *Suppose there is a vertex $w \in U \setminus U'$ with $L_w = \{c\}$, then we put w in U' and set $\Gamma'(w) = c$. If there is a vertex v where $L_v = \emptyset$, then we say* NO.

Lemma 8.7. *Reduction Rule 8.5 is safe.*

Proof. Γ must assign a colour to every vertex. If there is a vertex w with $L_w = \{c\}$, then we must set $\Gamma(w) = c$ for any satisfying assignment Γ. Thus, in the reduced instance, we fix the colouring of w by putting it in U' and setting $\Gamma'(w) = c$. In the reduced instance, U' is a superset of the U' in the original instance. Hence, a satisfying assignment Γ of the reduced instance is also a satisfying assignment of the original instance.

Similarly, by the correctness of the other Reduction Rules, if there is a vertex v where $L_v = \emptyset$, the current instance must be a NO instance. Thus we correctly say NO. Therefore, this Reduction Rule is safe. □

Given an instance $((U = U_1 \uplus U_2 \cdots U_p, \mathcal{F}'), \Psi_{\mathsf{family}}, \Psi_{\mathsf{parts}}, U', \Gamma', \{L_v \subseteq [r] | v \in U \setminus U'\})$ of Partitioned p-CFC, we apply Reduction Rules 8.2, 8.3, 8.4, 8.5 exhaustively. If in the process we infer that the given instance is a NO instance then we return the same. It could also happen that we get $\mathcal{F} = \emptyset$. In this case for every vertex $v \in U \setminus U'$, Γ assigns to v an element of $L(v)$ arbitrarily. Thus, from now onwards we assume that we neither conclude that the given instance is a NO instance nor obtain $\mathcal{F} = \emptyset$. We call an instance of Partitioned p-CFC *reduced* if Reduction Rules 8.2, 8.3, 8.4, 8.5 are not applicable. For simplicity, let $((U = U_1 \uplus U_2 \cdots U_p, \mathcal{F}'), \Psi_{\mathsf{family}}, \Psi_{\mathsf{parts}}, U', \Gamma', \{L_v \subseteq [r] | v \in U \setminus U'\})$ denote the reduced instance of Partitioned p-CFC. Observe that the reduced instance has the following properties:

1. For every vertex v, $|L_v| \geq 2$.
2. For every hyperedge h, $|\Gamma'^{-1}(\Psi_{\mathsf{family}}(h)) \cap h| = 0$.

We define the set $V_i \subseteq U \setminus U'$ as the set of vertices that have i in their list of admissible colours. Then there are two kinds of vertices in V_i: It could be that the vertex v has $i \in L_v$ and $\exists h \in \mathcal{F},\ v \in U_j \cap h$ such that $\Psi_{\mathsf{family}}(h) = i, \Psi_{\mathsf{parts}}(j) = i$. Or, the vertex v has $i \in L_v$. Also, for any h with $\Psi_{\mathsf{family}}(h) = i$, $v \notin h$.

To solve the reduced instance of Partitioned p-CFC, we will solve some r instances of an even more specialized problem that we define now. Let Partitioned UHS be the problem of determining, for a given partition $U_1 \uplus \ldots \uplus U_q$ of the universe and a family $\mathcal{F}$, whether there is a set S of vertices that uniquely hits all hyperedges of the input hypergraph (that is, for all $h \in \mathcal{F}$, $|h \cap S| = 1$) and where $\forall i \in [q],\ |U_i \cap S| = 1$. Now we define some sets based on $V_i \subseteq U$:

1. For every $j \in [r]$, and $x \in \Psi^{-1}_{\mathsf{parts}}(j)$ let $Z^x_j = U_x \cap V_j$ and $Z_j = \bigcup_{x \in \Psi^{-1}_{\mathsf{parts}}(j)} Z^x_j$.
2. For every $j \in [r]$, and $h \in \Psi^{-1}_{\mathsf{family}}(j)$ let $h_j = h \cap V_j$ and $\mathcal{F}_j = \{h_j \mid h \in \Psi^{-1}_{\mathsf{family}}(j)\}$.

Next we relate the instance of Partitioned p-CFC to Partitioned UHS.

Lemma 8.8. *Let* $((U = U_1 \uplus U_2 \cdots U_p, \mathcal{F}'), \Psi_{\mathsf{family}}, \Psi_{\mathsf{parts}}, U', \Gamma', \{L_v \subseteq [r] | v \in U \setminus U'\})$ *denote the reduced instance of* Partitioned p-CFC. *Then it is a* YES *instance of* Partitioned p-CFC *if and only if for all* $j \in [r]$, $(\uplus_{x \in \Psi^{-1}_{\mathsf{parts}}(j)} Z^x_j, \mathcal{F}_j)$ *is a* YES *instance of* Partitioned UHS.

Proof. First, suppose that $((U = U_1 \uplus U_2 \cdots U_p, \mathcal{F}'), \Psi_{\mathsf{family}}, \Psi_{\mathsf{parts}}, U', \Gamma', \{L_v \subseteq [r] | v \in U \setminus U'\})$ is a YES instance of Partitioned p-CFC. Then there is a satisfying assignment

Γ such that each hyperedge is conflict-free coloured, $\Gamma'(U') = \Gamma(U')$. For each $v \in U \setminus U', \Gamma(v) \in L_v$. Also, there exists a cfc-solution set $S = \{v_1, \ldots, v_p\}$ such that for all $i \in [p]$, $|S \cap U_i| = 1$. In the reduced instance, for all h, $|\Gamma'^{-1}(\Psi_{\mathsf{family}}(h)) \cap h| = 0$. Thus, $S \cap U' = \emptyset$. For each $i \in [r]$, we look at $S \cap V_i$. By definition of Z_i, every vertex in $S \cap V_i$ must belong to a part in Z_i. In particular, every vertex of $S \cap \Gamma^{-1}(i)$ must belong to a part in Z_i. Also, since every vertex of S belongs to a unique part of $U_1 \uplus U_2 \cdots U_p$, there is exactly one vertex in $S \cap Z_i^x$, for each $Z_i^x \in Z_i$. Also, we know that for every $h \in \mathcal{F}$, if $\mathsf{unicolelt}_S(h) = v_j$, then $\Gamma(v_j) = \Psi_{\mathsf{family}}(h)$. Thus, for each hyperedge $h \in \mathcal{F}_i$, $\mathsf{unicolelt}_S(h) \in S \cap \Gamma^{-1}(i)$. F or every other vertex $u \in h \setminus \mathsf{unicolelt}_S(h)$, $\Gamma(u) \neq i$ and therefore $u \notin S \cap \Gamma^{-1}(i)$. Thus, for every $i \in [r]$, $S_i = S \cap \Gamma^{-1}(i)$ is a unique hitting set of $\mathcal{F}_j$ with the property that $\forall x \in \Psi^{-1}_{\mathsf{parts}}(i)$, $|Z_i^x \cap S_i| = 1$. Thus, $(\uplus_{x \in \Psi^{-1}_{\mathsf{parts}}(j)} Z_j^x, \mathcal{F}_j)$ is a YES instance of Partitioned UHS.

In the reverse direction, suppose $(\uplus_{x \in \Psi^{-1}_{\mathsf{parts}}(j)} Z_j^x, \mathcal{F}_j)$ is a YES instance of Partitioned UHS. Then a solution set S_i is a unique hitting set of $\mathcal{F}_j$ with the property that $\forall x \in \Psi^{-1}_{\mathsf{parts}}(i)$, $|Z_i^x \cap S_i| = 1$. By definition, $S_i \subseteq Z_i \subseteq V_i$. First, for each vertex $v \in S_i$, we assign $\Gamma(v) = i$. For each $w \in U'$, we must set $\Gamma(w) = \Gamma'(w)$. Now, we look at a vertex $w \in V_i \setminus S_i$. Look at the colours in $L_w \setminus \{i\}$. In the reduced instance, it must be the case that, for any h with $\Psi_{\mathsf{family}}(h) \neq i$, $w \notin h$. For a vertex $w \in (U \setminus U') \setminus \bigcup_{j \in [r]} S_i$, we arbitrarily pick a colour $c \in L_w \setminus \{i\}$ and set $\Gamma(w) = c$. Every hyperedge h has exactly one vertex in the colour class $\Psi_{\mathsf{family}}(h)$, namely the vertex in $S_{\Psi_{\mathsf{family}}(h)} \cap h$ that uniquely hits h. Thus, Γ is a satisfying assignment and $((U = U_1 \uplus U_2 \cdots U_p, \mathcal{F}'), \Psi_{\mathsf{family}}, \Psi_{\mathsf{parts}}, U', \Gamma', \{L_v \; subseteq[r] | v \in U \setminus U'\})$ is a YES instance of Partitioned p-CFC. □

Lemma 8.8 allows us to reduce an instance of the Partitioned p-CFC problem to r instances of Partitioned UHS. Next, we design an algorithm for Partitioned UHS.

Lemma 8.9. Partitioned UHS*, where the number of hyperedges is m, the universe size is n and a $q \leq m$ partitioning of the universe is given, is* FPT *parameterized by m. The running time of the algorithm is $4^m \cdot (n+m)^{\mathcal{O}(1)}$.*

Proof. We are given as input a hypergraph $H = (U, \mathcal{F})$ and a q-partition $U = (U_1, \ldots, U_q)$ of the universe. We define a function $\mathcal{A}$ that takes as input a pair $(\mathcal{E}, C)$, where $\mathcal{E} \subseteq \mathcal{F}$ and $C \subseteq [q]$. The function outputs 1 if the subfamily $\mathcal{E}$ can be uniquely hit by a set S, where $\forall i \in C$, $|U_i \cap S| = 1\}$, and 0 otherwise. Define for each vertex v that belongs to a part $U_i, i \in C$, a subfamily $\mathcal{E}_v = \{h \mid v \in h\}$. We define the function $\mathcal{A}$ using the following recurrence relation:

$$\mathcal{A}(\mathcal{E}, C) = \begin{cases} \max\limits_{c \in C, v \in U_c} \mathcal{A}(\mathcal{E} \setminus \mathcal{E}_v, C \setminus \{c\}), & \text{if } \mathcal{E} \neq \emptyset \\ 1, & \text{if } \mathcal{E} = \emptyset \text{ and } C = \emptyset \\ 0, & \text{if } \mathcal{E} = \emptyset \\ 0, & \text{if } C = \emptyset \end{cases}$$

We prove the correctness of this recurrence by induction on the size of the set $C \subseteq [q]$. In the base case, when $\mathcal{E} = \emptyset, C = \emptyset$, then trivially this family has been uniquely hit and therefore $\mathcal{A}(\mathcal{E}, C) = 1$. When $C = \emptyset$, then no subfamily can be uniquely hit and therefore $\mathcal{A}(\mathcal{E}, C) = 0$ for any subfamily $\mathcal{E}$. When $\mathcal{E} = \emptyset$, but $C \neq \emptyset$, then the family cannot have a unique hitting set S with the property $\forall i \in C$, $|U_i \cap S| = 1\}$. Therefore, $\mathcal{A}(\mathcal{E}, C) = 0$ for

any set $C \neq \emptyset$. Now, let $|C| \geq 1$. Suppose we have correctly calculated $\mathcal{A}(\mathcal{E}', C')$ for all pairs $(\mathcal{E}', C')$ where $|C'| < |C|$ and $\mathcal{E}' \subseteq \mathcal{F}$. There can be two cases:

1. Suppose $\mathcal{A}(\mathcal{E}, C) = 1$. Then there is a solution set S such that $\forall i \in C,\ |U_i \cap S| = 1\}$. Take one $i \in C$ and let $v_i \in S \cap U_i$. Then $S - \{v_i\}$ uniquely hits $\mathcal{E} \setminus \mathcal{E}_{v_i}$ and $\forall j \in C \setminus \{i\},\ |U_j \cap S| = 1\}$. Then, by induction hypothesis, $\mathcal{A}(\mathcal{E} \setminus \mathcal{E}_{v_i}, C \setminus \{i\}) = 1$. Hence, we correctly calculate $\mathcal{A}(\mathcal{E}, C)$.

2. On the other hand, suppose $\mathcal{A}(\mathcal{E}, C) = 0$. Then, there was no such unique hitting set S such that $\forall i \in C,\ |U_i \cap S| = 1\}$. If there is a vertex $v \in U_j, j \in C$ such that $\mathcal{A}(\mathcal{E} \setminus \mathcal{E}_v, C \setminus \{j\}) = 1$, then, by induction hypothesis, there is a solution set S' for $\mathcal{E} \setminus \mathcal{E}_v$ such that $\forall i \in C \setminus \{j\},\ |U_i \cap S| = 1\}$. But then $S \cup \{v\}$ is a solution set for $\mathcal{E}$ such that $\forall i \in C,\ |U_i \cap S| = 1\}$. This is a contradiction.

Thus the recurrence is correct.

It is enough to solve this recurrence for every pair $(\mathcal{E}, C)$. The given instance is a YES instance of Partitioned UHS if $\mathcal{A}(\mathcal{F}, [q]) = 1$. We look up the value of one subproblem in order to calculate the function $\mathcal{A}(\mathcal{E}, C)$ and there are 4^m such pairs $(\mathcal{E}, C)$. Thus, the running time for solving the recurrence is $4^m(n+m)^{\mathcal{O}(1)}$. □

Lemmata 8.8, 8.9 and safeness of the Reduction Rules 8.2, 8.3, 8.4, 8.5 together result in the following algorithm for Partitioned p-CFC.

Lemma 8.10. Partitioned p-CFC *can be solved in time* $2^{p+|\mathcal{F}|} \cdot N^{\mathcal{O}(1)}$.

Next, using Lemma 8.10 and the method of colour coding technique of [Alon 1995] we give an algorithm for p-CFC. Towards this we need the following notion of a Perfect Hash Family. A Perfect Hash Family is a family of functions, whose domain is a universe U of n elements and range is a set of k elements, and with the following property: for every k-sized subset $S \subseteq U$, there is a function ζ in the family that maps S to the range injectively. That is, every element of S maps to a different number in $[k]$. The following Proposition shows that such families are constructive [Naor 1995].

Proposition 8.5. *For any* n *and* $k \leq n$, *a* (n, k)*-Perfect Hash Family of size* $e^k k^{\mathcal{O}(\log k)} \log n$ *can be deterministically computed in time* $e^k k^{\mathcal{O}(\log k)} n \log n$.

Our main theorem is the following.

Theorem 8.2. p-CFC *can be solved in time* $2^{\mathcal{O}(k \log\log k + k \log r)} \cdot N^{\mathcal{O}(1)}$.

Proof. Let $((U, \mathcal{F}), k, r)$ be an instance of p-CFC. Recall that $|U| = n$, $|\mathcal{F}| = m$ and $N = n+m$. Given an instance we first apply Theorem 8.1 and obtain an equivalent instance with at most 4^k vertices and $\mathcal{O}(k \log k)$ hyperedges. We run through all $p \leq k$. Since the number of hyperedges in the input instance is $\alpha_{uc} k \log k$, the number of subfamilies of size k is $\binom{\alpha_{uc} k \log k}{k} \leq \frac{(\alpha_{uc} k \log ke)}{k}^k \leq (\alpha_{uc} \log k)^k$. We guess a subfamily $\mathcal{F}'$ of hyperedges that will be conflict-free coloured. That is, we are trying to find a colouring $\Gamma : U \to [r]$ such that each hyperedge h in $\mathcal{F}'$ is conflict-free coloured. Let S be a hypothetical cfc-solution corresponding to it. In other words, for each hyperedge h in $\mathcal{F}'$, a uniquely coloured vertex of h (with respect to Γ) belongs to S. We guess the size of $|S|$, say $p \leq k$. For a fixed p, let

$\mathfrak{F}$ be the family of (n,p)-Perfect Hash Family of size $e^p p^{\mathcal{O}(\log p)} \log n$. By the property of $\mathfrak{F}$, we know that there exists a function $\zeta \in \mathfrak{F}$ that maps S to $[p]$ injectively. Let $U_1, \ldots, U_p$ denote the partition of U given by ζ. Observe that after this we will be seeking for a cfc-solution S such that $|S \cap U_i| = 1$ for all $i \in [p]$.

Next for each hyperedge h in $\mathcal{F}'$, we guess the colour of a vertex in h that is uniquely coloured by Γ. There are r^k such guesses. Thus, after this guess, we define a function $\Psi_{\mathsf{family}} : \mathcal{F}' \to [r]$ such that h is assigned the colour of the vertex in h that will be uniquely coloured by Γ. Finally, for the potential solution set S we guess the colour of each vertex given by Γ. Since we are looking for a cfc-solution set S, such that $\forall i \in [p],\ |U_i \cap S| = 1\}$ it is equivalent to say that we guess an r partitioning of the p parts in $U = (U_1, \ldots, U_p)$. That is, the vertex of $S \in U_i$ will be assigned to each colour by Γ. To express this guess, we define another function $\Psi_{\mathsf{parts}} : [p] \to [r]$ such that $\Psi_{\mathsf{parts}}(j) = i$ if the vertex x in $S \cap U_j$ will have $\Gamma(x) = i$. Thus, there are r^p guesses for the colouring of the potential solution set S by Γ. At the end of this sequence of guesses, we have fixed a choice of hyperedges that are to be r conflict-free coloured, a colouring of the potential solution set S (without actually knowing the vertices of S, this essentially means a partitioning of the parts of U) and a partitioning of the hyperedges according to which colour of Γ will determine that the hyperedge is conflict-free coloured. This results in the following instance of Partitioned p-CFC: $((U = U_1 \uplus U_2 \cdots U_p, \mathcal{F}'), \Psi_{\mathsf{family}}, \Psi_{\mathsf{parts}}, U' = \emptyset, (\forall v \in U : L_v = [r]))$. By Lemma 8.10 we know that we can solve this in time $2^{p+k} \cdot N^{\mathcal{O}(1)} \leq 4^k \cdot N^{\mathcal{O}(1)}$. Thus the overall running time for p-CFC is upper bounded by the number of guesses and the running time of an algorithm for Partitioned p-CFC. Thus, the running time of the algorithm is upper bounded by:

$$\binom{\alpha_{uc} k \log k}{k} \times k \times |\mathfrak{F}| \times r^k \times r^k \times 4^k \cdot N^{\mathcal{O}(1)} = 2^{\mathcal{O}(k \log\log k + k \log r)} \cdot N^{\mathcal{O}(1)}.$$

This concludes the proof. □

8.4 FPT Algorithm for p-UMC

In this section, we give the full description of an FPT algorithm for p-UMC that runs in $2^{\mathcal{O}(k \log\log k + k \log r)} \cdot N^{\mathcal{O}(1)}$ time. Here, $N = |U| + |\mathcal{F}|$. The algorithm is very similar to that given for p-CFC.

The results of Lemma 8.3 and Theorem 8.1 can be modified for p-UMC.

Theorem 8.3. p-UMC *has a kernel with at most* 4^k *vertices and* $\mathcal{O}(k \log k)$ *sets.*

Therefore, we assume that our input instance for sc p-UMC contains at most $\mathcal{O}(k \log k)$ hyperedges and 4^k vertices.

Given a set $S \subseteq U$, a subfamily $\mathcal{F}'$, and a colouring $\Gamma : U \to [r]$, we say that S is a *um-solution* if each hyperedge h in $\mathcal{F}'$ is unique-maximum coloured and the vertex, which is unique-maximum coloured in h, belongs to S. Furthermore, given such a set S and a hyperedge h, let $\mathsf{unicolelt}_S(h)$ denote the unique-maximum coloured vertex of h, that belongs to S. Our strategy for the FPT algorithm is same as in the case of p-CFC. We define an auxiliary problem and give an FPT algorithm for this problem. Finally, we reduce our problem to this one with some guesses and by using the colour coding technique,

introduced by Alon et. al. in [Alon 1995]. Thus we obtain the desired algorithm for p-UMC.

PARTITIONED p-UMC	**Parameter:** $r + p + \lvert\mathcal{F}\rvert$
Input: A hypergraph $(U = U_1 \uplus U_2 \cdots U_p, \mathcal{F})$, a function $\Psi_{\mathsf{family}} : \mathcal{F} \to [r]$, $\Psi_{\mathsf{parts}} : [p] \to [r]$, a subset $U' \subseteq U$ and a colouring function $\Gamma' : U' \to [r]$, for every $v \in U \setminus U'$, a list $L_v \subseteq [r]$	
Question: Does there exist a colouring function $\Gamma : U \to [r]$ such that: Each hyperedge is unique-maximum coloured, $\Gamma(U') = \Gamma'(U')$. For each $v \in U \setminus U', \Gamma(v) \in L_v$. Also, the um-solution set S, defined by Γ , is of size exactly p. For all $i \in [p]$, $\lvert S \cap U_i\rvert = 1$ and for every $h \in \mathcal{F}$, $\mathsf{unicolelt}_S(h) \in \bigcup_{j \in \Psi^{-1}_{\mathsf{parts}}(\Psi_{\mathsf{family}}(h))} U_j$?	

In simple words, the problem definition can be explained as follows. We are given a partitioning of the universe U into p-parts and a partial colouring function Γ' on a subset U'. We are looking for a colouring $\Gamma : U \to [r]$ which extends Γ'. Each vertex v in $U \setminus U'$ has a list of admissible colours, and Γ must choose a colour from L_v. Also, due to Γ, each hyperedge is unique-maximum coloured and the um-solution set S, due to Γ, is such that it contains exactly one vertex from each part. Suppose the hypothetical set S is $\{x_1, x_2, \ldots, x_p\}$ (think of x_i as a variable) where $x_i \in U_i$. The function Ψ_{parts} is used to guess the colour of x_i in Γ. The function Ψ_{family} divides the family $\mathcal{F}$ into r *chunks* (not to be confused with parts and colouring). The idea is that the unique-maximum coloured vertex of $h \in \mathcal{F}$, say x_j, has been assigned the same colour by Γ as h has been assigned to the chunk number by Ψ_{family}, i.e, $\Gamma(x_j) = \Psi_{\mathsf{family}}(h)$. Next we show how we can solve the PARTITIONED p-UMC problem.

Given an instance $((U = U_1 \uplus U_2 \cdots U_p, \mathcal{F}'), \Psi_{\mathsf{family}}, \Psi_{\mathsf{parts}}, U', \Gamma', \{L_v \subseteq [r] | v \in U \setminus U'\})$ of PARTITIONED p-UMC, we first do a polynomial time preprocessing of the instance. For all $v \in U'$, we must set $\Gamma(v) = \Gamma'(v)$. In the following Reduction Rules, we show that the input functions Ψ_{family} and Ψ_{parts} allow us to prune the list of some of the vertices. The first reduction rule is necessary for a unique-maximum colouring that supports the function Ψ_{family}.

Reduction Rule 8.6. *For each hyperedge h, and each vertex $v \in h$, remove the colours $r \leq i > \Psi_{\mathsf{family}}(h)$ from $L(v)$.*

Lemma 8.11. *Reduction Rule 8.6 is safe.*

Proof. The required unique-maximum colouring Γ should be such that for each hyperedge h, the unique-maximum coloured vertex of h must receive the same colour as $\Psi_{\mathsf{family}}(h)$. Therefore, no vertex of h can be given a colour which is higher in order than $\Psi_{\mathsf{family}}(h)$. This implies that the Reduction Rule is safe. □

The next rule also ensures the properties of unique-maximum colouring.

Reduction Rule 8.7. *Given a hyperedge h, if there is a vertex $w \in h \cap U'$ such that $\Gamma'(w) > \Psi_{\mathsf{family}}(h)$, then we say* NO.

Lemma 8.12. *Reduction Rule 8.7 is safe.*

Proof. The definition of the problem requires Γ to be an extension of Γ'. It also requires the unique-maximum coloured vertex of each hyperedge h to be coloured by $\Psi_{\mathsf{family}}(h)$.

By definition of unique-maximum colouring, all other vertices of h must receive a colour of lower order than $\Psi_{\mathsf{family}}(h)$. Therefore, for a YES instance, it must be the case that for each vertex w in $h \cap U'$, $\Gamma'(w) \leq \Psi_{\mathsf{family}}(h)$. This implies the correctness of the Reduction Rule. □

The next few reduction rules are similar to the rules described for the FPT algorithm of p-UMC.

Reduction Rule 8.8. *Suppose there is a hyperedge h containing a unique vertex $w \in U'$ such that $\Psi_{\mathsf{family}}(h) = \Gamma'(w)$. Then, for every $v \in h \setminus \{w\}$ we delete $\Psi_{\mathsf{family}}(h)$ from L_v. We delete h from $\mathcal{F}$.*

The proof of correctness for this rule is very similar to that of Reduction Rule 8.2.

Reduction Rule 8.9. *If there is a vertex $v \in U_i, i \in [p]$, and $h \in \mathcal{F}$, such that $v \in h$, $\Psi_{\mathsf{family}}(h) \neq \Psi_{\mathsf{parts}}(i)$, then we remove the colour $\Psi_{\mathsf{family}}(h)$ from the list of v.*

This proof of correctness is similar to that of Reduction Rule 8.3.

Reduction Rule 8.10. *If there are two vertices $v, w \in U'$ and a hyperedge $h \in \mathcal{F}$, such that $\Psi_{\mathsf{family}}(h) = \Gamma'(v) = \Gamma'(w)$, then we say* NO.

The proof of correctness for this Reduction Rule is similar to Reduction Rule 8.4.

Reduction Rule 8.11. *Suppose there is a vertex $w \in U \setminus U'$ with $L_w = \{c\}$, then we put w in U' and set $\Gamma'(w) = c$. If there is a vertex v where $L_v = \emptyset$, then we say* NO.

This follows from the safeness of Reduction Rule 8.5.

Given an instance $((U = U_1 \uplus U_2 \cdots U_p, \mathcal{F}'), \Psi_{\mathsf{family}}, \Psi_{\mathsf{parts}}, U', \Gamma', \{L_v \subseteq [r] | v \in U \setminus U'\})$ of Partitioned p-UMC, we apply the above Reduction Rules exhaustively. If in the process we infer that the given instance is a NO instance then we return the same. It could also happen that we get $\mathcal{F} = \emptyset$. In this case, for every vertex $v \in U \setminus U'$, Γ assigns to v an element of $L(v)$ arbitrarily. Thus, from now onwards we assume that we neither conclude that the given instance is a NO instance nor obtain $\mathcal{F} = \emptyset$. We call an instance of Partitioned p-UMC *reduced* if the above Reduction Rules are not applicable. For simplicity, let $((U = U_1 \uplus U_2 \cdots U_p, \mathcal{F}'), \Psi_{\mathsf{family}}, \Psi_{\mathsf{parts}}, U', \Gamma', \{L_v \subseteq [r] | v \in U \setminus U'\})$ denote the reduced instance of Partitioned p-UMC. Observe that the reduced instance has the following properties:

1. For every vertex v, $|L_v| \geq 2$.
2. For every hyperedge h, $|\Gamma'^{-1}(\Psi_{\mathsf{family}}(h)) \cap h| = 0$.
3. For every hyperedge h, and each vertex $v \in h$, $L(v)$ contains colours that are of order at most $\Psi_{\mathsf{family}}(h)$.

We define the set $V_i \subseteq U \setminus U'$ as the set of vertices that have i in their list of admissible colours. Then there are two kinds of vertices in V_i: It could be that the vertex v has $i \in L_v$ and $\exists h \in \mathcal{F},\ v \in U_j \cap h$ such that $\Psi_{\mathsf{family}}(h) = i, \Psi_{\mathsf{parts}}(j) = i$. Or, the vertex v has

$i \in L_v$. Moreover, for any h with $\Psi_{\mathsf{family}}(h) = i$, $v \notin h$. Also, for each h that contains v, $\Psi_{\mathsf{family}}(h) > i$.

To solve the reduced instance of Partitioned p-UMC, we will again solve r instances of Partitioned UHS. We define some sets based on $V_i \subseteq U$:

1. For every $j \in [r]$, and $x \in \Psi^{-1}_{\mathsf{parts}}(j)$ let $Z^x_j = U_x \cap V_j$ and $Z_j = \bigcup_{x \in \Psi^{-1}_{\mathsf{parts}}(j)} Z^x_j$.
2. For every $j \in [r]$, and $h \in \Psi^{-1}_{\mathsf{family}}(j)$ let $h_j = h \cap V_j$ and $\mathcal{F}_j = \{h_j \mid h \in \Psi^{-1}_{\mathsf{family}}(j)\}$.

Next we relate the instance of Partitioned p-CFC to Partitioned UHS.

Lemma 8.13. *Let* $((U = U_1 \uplus U_2 \cdots U_p, \mathcal{F}'), \Psi_{\mathsf{family}}, \Psi_{\mathsf{parts}}, U', \Gamma', \{L_v \subseteq [r] | v \in U \setminus U'\})$ *denote the reduced instance of* Partitioned p-UMC. *Then it is a* YES *instance of* Partitioned p-UMC *if and only if for all* $j \in [r]$, $(\uplus_{x \in \Psi^{-1}_{\mathsf{parts}}(j)} Z^x_j, \mathcal{F}_j)$ *is a* YES *instance of* Partitioned UHS.

This proof is very similar to the proof of Lemma 8.8. Lemmata 8.13 and 8.9 together result in the following algorithm for Partitioned p-CFC.

Lemma 8.14. Partitioned p-UMC *can be solved in time* $2^{p+|\mathcal{F}|} \cdot N^{\mathcal{O}(1)}$.

Next, using Lemma 8.14 and the method of colour coding technique of [Alon 1995] we give an algorithm for p-UMC.

Theorem 8.4. p-UMC *can be solved in time* $2^{\mathcal{O}(k \log\log k + k \log r)} \cdot N^{\mathcal{O}(1)}$.

Proof. Let $((U, \mathcal{F}), k, r)$ be an instance of p-UMC. Recall that $|U| = n$, $|\mathcal{F}| = m$ and $N = n+m$. Given an instance we first apply Theorem 8.3 and obtain an equivalent instance with at most 4^k vertices and $\mathcal{O}(k \log k)$ hyperedges. We run through all $p \leq k$. Since the number of hyperedges in the input instance is $\alpha_{uc} k \log k$, the number of subfamilies of size k is $\binom{\alpha_{uc} k \log k}{k} \leq \frac{(\alpha_{uc} k \log ke)}{k}^k \leq (\alpha_{uc} \log k)^k$. We guess a subfamily $\mathcal{F}'$ of hyperedges that will be unique-maximum coloured. That is, we are trying to find a colouring $\Gamma : U \to [r]$ such that each hyperedge h in $\mathcal{F}'$ is unique-maximum coloured. Let S be the hypothetical um-solution corresponding to it. In other words, for each hyperedge h in $\mathcal{F}'$, the unique-maximum coloured vertex of h (with respect to Γ) belongs to S. We guess the size of $|S|$, say $p \leq k$. For a fixed p, let $\mathfrak{F}$ be the family of (n,p)-Perfect Hash Family of size $e^p p^{\mathcal{O}(\log p)} \log n$. By the property of $\mathfrak{F}$, we know that there exists a function $\zeta \in \mathfrak{F}$ that maps S to $[p]$ injectively. Let $U_1, \ldots, U_p$ denote the partition of U given by ζ. Observe that after this we will be seeking for the um-solution S with the property that $|S \cap U_i| = 1$ for all $i \in [p]$.

Next for each hyperedge h in $\mathcal{F}'$, we guess the colour of the vertex in h that is unique-maximum coloured by Γ. There are r^k such guesses. Thus, after this guess, we define a function $\Psi_{\mathsf{family}} : \mathcal{F}' \to [r]$ such that h is assigned the colour of the vertex in h that will be unique-maximum coloured by Γ. Finally, for the potential solution set S we guess the colour of each vertex given by Γ. Since we are looking for a um-solution set S, such that $\forall i \in [p],\ |U_i \cap S| = 1\}$ it is equivalent to say that we guess an r partitioning of the p parts in $U = (U_1, \ldots, U_p)$. That is, the vertex of $S \in U_i$ will be assigned a colour by Γ. To express this guess, we define another function $\Psi_{\mathsf{parts}} : [p] \to [r]$ such that

$\Psi_{\mathsf{parts}}(j) = i$ if the vertex x in $S \cap U_j$ will have $\Gamma(x) = i$. Thus, there are r^p guesses for the colouring of the potential solution set S by Γ. At the end of this sequence of guesses, we have fixed a choice of hyperedges that are to be r unique-maximum coloured, a colouring of the potential solution set S (without actually knowing the vertices of S, this essentially means a partitioning of the parts of U) and a partitioning of the hyperedges according to which colour of Γ will determine that the hyperedge is unique-maximum coloured. This results in the following instance of Partitioned p-UMC: $((U = U_1 \uplus U_2 \cdots U_p, \mathcal{F}'), \Psi_{\mathsf{family}}, \Psi_{\mathsf{parts}}, U' = \emptyset, (\forall v \in U : L_v = [r]))$. By Lemma 8.14 we know that we can solve this in time $2^{p+k} \cdot N^{\mathcal{O}(1)} \leq 4^k \cdot N^{\mathcal{O}(1)}$. Thus the overall running time for p-UMC is upper bounded by the number of guesses and the running time of an algorithm for Partitioned p-UMC. Thus, the running time of the algorithm is upper bounded by:

$$\binom{\alpha_{uc} k \log k}{k} \times k \times |\mathfrak{F}| \times r^k \times r^k \times 4^k \cdot N^{\mathcal{O}(1)} = 2^{\mathcal{O}(k \log\log k + k \log r)} \cdot N^{\mathcal{O}(1)}.$$

This concludes the proof. □

8.5 Exact Algorithm for Max-Conflict Free Colouring

In this section, we give an exact algorithm for solving Max-CFC for hypergraphs. We give a recurrence on subproblems, using which we can give a dynamic programming algorithm to solve the problem. However, a much faster algorithm can be designed using the technique of subset convolutions on functions.

Theorem 8.5. Max-CFC *for hypergraphs can be solved by an exact algorithm that runs in* $\mathcal{O}(2^{(m+n)})$ *time.*

Proof. Let $H = (U, \mathcal{F})$ be the input hypergraph. Suppose, for a given hypergraph, there is a procedure to decide whether there exists an r-colouring that is conflict-free. Then, we can generate all subsets $\mathcal{F}'$ of $\mathcal{F}$, such that there exists an r-colouring of vertices of $(U(\mathcal{F}'), \mathcal{F}')$ that is conflict-free, by running this procedure for all subsets . Then solving the Max-CFC problem reduces to picking the maximum sized subsets among those.

We now give a procedure to find the minimum number of colours required to conflict-free colour a given hypergraph, $(U', \mathcal{F}')$. Let χ' be a r-colouring on U' and let $\mathcal{F}'$ be conflict-free coloured by χ'. Then χ' partitions U' into r partitions, $U_1, U_2, \ldots, U_r$, such that the following property is true.

$$\forall F \in \mathcal{F}', \exists i \in [r] \text{ such that } |F \cap U_i| = 1.$$

Let $\mathcal{F}_1$ be the set of hyperedges such that $\forall F \in \mathcal{F}_1, |F \cap U_1| = 1$. In other words, all the hyperedges in $\mathcal{F}_1$ have a unique vertex coloured by colour 1. Then, if we correctly guessed U_1 then solving whether $\mathcal{F}'$ has an r conflict-free colouring in U is equivalent to solving the subproblem of whether $\mathcal{F}' \setminus \mathcal{F}_1$ has an $r-1$ conflict-free colouring in $U \setminus U_1$.

Let $\mathcal{C}(X, \mathcal{E})$ be the minimum number of colours needed to conflict-free colour the hypergraph $(X, \mathcal{E})$. We give the following recurrence relation to find $\mathcal{C}(X, \mathcal{E})$.

$$\mathcal{C}(X, \mathcal{E}) = \begin{cases} min_{X' \subseteq X : \exists h \in \mathcal{E}, |h \cap X'| = 1} \{1 + \mathcal{C}(X \setminus X', \mathcal{E} \setminus \mathcal{E}')\}, & \text{if } X \neq \phi \\ 0, & \text{if } X = \phi \end{cases} \tag{8.1}$$

where $\mathcal{E}' = \{h \in \mathcal{E} || h \cap X'| = 1\}$.

We prove the correctness of the recurrence by induction on the size of X. When $|X| = 0$, the recurrence correctly returns 0.

Now assume $|X| > 0$. Assume $X' \subseteq X$ is a colour class of a conflict-free colouring that uses $\chi_{cf}((X, \mathcal{E}|_X))$ colours. Then X' uniquely colours all hyperedges that contain exactly one element from X'. $\mathcal{E}'$ represents the family of this hyperedges. The remaining hyperedges $\mathcal{E} \setminus \mathcal{E}'$ need to be uniquely covered by colour classes in $X \setminus X'$. By induction hypothesis, $\mathcal{C}(X \setminus X', \mathcal{E} \setminus \mathcal{E}') = \chi_{cf}((X \setminus X', \mathcal{E} \setminus \mathcal{E}'|_{X \setminus X'}))$. Hence, $\mathcal{C}(X \setminus X', \mathcal{E} \setminus \mathcal{E}') + 1$ returns the the value of $\chi_{cf}((X, \mathcal{E}|_X))$. Since the recurrence considers all possible subsets of X, one of them is the correct guess for X' and returns the minimum value.

We can compute the function $\mathcal{C}$ in time $\mathcal{O}(2^m \cdot \Sigma_{0 \leq i \leq n} \binom{n}{i} 2^i) = \mathcal{O}(3^n 2^m)$. Once this is done, finding a largest subfamily $\mathcal{F}' \subseteq \mathcal{F}$, such that $(U(\mathcal{F}'), \mathcal{F}')$ can be r conflict-free coloured, can be done in $\mathcal{O}(2^m)$ time. Hence, we could solve the Max-CFC problem in $\mathcal{O}(3^n \cdot 2^m)$ time. However, we can improve the running time by quite a bit.

Let us relax the definition of $\mathcal{C}$ to be the function which takes a pair $(U', \mathcal{F}')$, where $U' \subseteq U, \mathcal{F}' \subseteq \mathcal{F}$, and, when $\chi_{cf}((U', \mathcal{F}'|_{U'})) \leq r$, correctly maps it to $\chi_{cf}((U', \mathcal{F}'|_{U'}))$. Otherwise, it could map $(U', \mathcal{F}')$ to some value between $n+1$ and $r(n+1)$, thereby clearly indicating that $(U', \mathcal{F}'|_{U'})$ is not r conflict-free colourable. Then, too we can identify subfamilies $\mathcal{F}'$ such that $\mathcal{C}((U(\mathcal{F}'), \mathcal{F}')) \leq r$ and pick one subfamily which has that largest size. From now on, by $\mathcal{C}$, we will refer to this new definition of $\mathcal{C}$.

To facilitate the calculation of this newly defined $\mathcal{C}$, we define another function f which takes as input a pair $(X \subseteq U, \mathcal{E} \subseteq \mathcal{F})$. When $X = \emptyset$, for any subfamily $\mathcal{E}$ we define the function $f(X, \{\mathcal{E}\}) = 0$. For each nonempty $X \subseteq U$ and $\mathcal{E} \subseteq \mathcal{F}$, $f(X, \mathcal{E}) = 1$ if for each $H \in \mathcal{E}, |H \cap X| = 1$. Otherwise, $f(X, \mathcal{F}') = n + 1$. Notice that it takes $\mathcal{O}(2^{(n+m)}) \cdot n^{\mathcal{O}(1)}$ time to calculate the function f. Using this function f, we are ready to define the function $\mathcal{C}(X, \mathcal{E})$ as follows:

$$\mathcal{C}(X, \mathcal{E}) = \min_{X_1 \uplus X_2 \uplus \ldots \uplus X_r = X; \mathcal{E}_1 \uplus \ldots \uplus \mathcal{E}_r = \mathcal{E}} f(X_1, \mathcal{E}_1) + \ldots f(X_r, \mathcal{E}_r) \tag{8.2}$$

Finally, we identify a subfamily $\mathcal{F}'$ of $\mathcal{F}$, of largest size, such that $\mathcal{C}(U(\mathcal{F}'), \mathcal{F}')$ is r conflict-free colourable.

Correctness

The correctness for the procedure described above is similar to the previous arguments. First, we show that, when X is nonempty, the function $\mathcal{C}(X, \mathcal{E})$ determines whether $(X, \mathcal{E}|_X)$ can be r conflict-free coloured or not. Also, when the tuple is r conflict-free colourable, the function returns the minimum number of colours required. We prove the correctness by case analysis of X. When $|X| = 1$, then for any subfamily $\mathcal{F}'$, $f(X, \mathcal{F}') = 1$ if and only if every hyperedge in $\mathcal{F}'$ contains the vertex of X. Thus, the hypothesis is true for the base case.

Now assume $|X| > 1$. First, suppose $\chi_{cf}((X, \mathcal{E}|_X)) \leq r$. Assume $(X_1 \uplus X_2 \uplus \ldots \uplus X_r)$ is a conflict-free colouring that realizes $\chi_{cf}((X, \mathcal{E}|_X)) \leq r$. Some of the X_i's could be emptysets if $\chi_{cf}((X, \mathcal{E}|_X)) < r$. Each nonempty X_i uniquely colours all hyperedges that contain exactly one element from X_i. Without loss of generality, we may assume that X_1 is nonempty.

Let $\mathcal{E}_1 \subseteq \mathcal{E}$ represent the subfamily of hyperedges that contain exactly one element from X_1. For $i > 1$, if X_i is empty, then $\mathcal{E}_i = \emptyset$. When X_i is nonempty, let $\mathcal{E}_i \subseteq \mathcal{E} - \bigcup_{1 \leq j \leq i-1} \mathcal{E}_j$ be the subfamily of hyperedges that contain exactly one element from X_i. Notice that for all $1 \leq i \leq r$, when X_i is nonempty, $f(X_i, \mathcal{E}_i) = 1$. When $X_i = \emptyset$, from the definition of f, $f(X_i, \mathcal{E}_i) = 0$. Hence, $\mathcal{C}(X, \mathcal{E}) = \min_{X_1 \uplus X_2 \uplus \ldots \uplus X_r = X; \mathcal{E}_1 \uplus \ldots \uplus \mathcal{E}_r = \mathcal{E}} f(X_1, \mathcal{E}_1) + \ldots f(X_r, \mathcal{E}_r)$ returns the minimum number of colours required to conflict-free colour $(X, \mathcal{E}|_X)$.

On the other hand, if $\mathcal{C} > r$, then for any r-partition $(X_1 \uplus X_2 \uplus \ldots \uplus X_r)$ of X, and r-partition $\mathcal{E}_1 \uplus \ldots \uplus \mathcal{E}_r$ of $\mathcal{E}$, there will be at least one tuple $(X_i, \mathcal{E}_i)$ such that $f(X_i, \mathcal{E}_i) = n+1$. Therefore, we correctly calculate that $\mathcal{C}(X, \mathcal{E}) > r$. Notice that $(n+1) \leq \mathcal{C}(X, \mathcal{E}) \leq r(n+1)$.

A subfamily $\mathcal{F}'$ which is r conflict-free colourable and of largest size should have $\mathcal{C}(U(\mathcal{F}'), \mathcal{F}') \leq r$. Thus, we go through all such tuples corresponding to subfamilies, and determine a largest subfamily that is r conflict-free colourable.

Running Time

The first step of the algorithm is to compute the function $\mathcal{C} : 2^U \times 2^{\mathcal{F}} \to \mathbb{Z} \cup \{\infty\}$ for all subsets of U and all subfamilies of $\mathcal{F}$. As stated earlier, it takes $\mathcal{O}(2^{(n+m)})$ time to calculate the function $f : 2^U \times 2^{\mathcal{F}} \to \mathbb{Z} \cup \{\infty\}$. The definition of $\mathcal{C}$ involves taking the minimum over the sum of r values from the range of f. The range of f is bounded between 0 and $n+1$, which makes the range of $\mathcal{C}$ bounded between 0 and $r(n+1)$. Using Proposition 8.2, computing the function $\mathcal{C}$ is equivalent to computing a sequence of $\log r$ subset convolutions over the integer min-sum semi-ring in the following way: We first calculate the subset convolution $f * f$ and obtain a function $g_1 : 2^U \times 2^{\mathcal{F}} \to \mathbb{Z} \cup \{\infty\}$, then we compute the subset convolution $g_1 * g_1$ to obtain a function $g_2 : 2^U \times 2^{\mathcal{F}} \to \mathbb{Z} \cup \{\infty\}$ and so on for $\log r$ steps. Hence, the algorithm for computing the function $\mathcal{C}$ runs in $\mathcal{O}(2^{n+m})$ time.

Finally, the algorithm runs through all subfamilies $\mathcal{F}'$ and finds out the largest sized subfamily such that $(U(\mathcal{F}'), \mathcal{F}')$ is r conflict-free coloured. Thus the total running time of the algorithm is $\mathcal{O}(2^{n+m})$. □

It is to be noted that by setting $r = n$, $\mathcal{C}(U, \mathcal{F})$ returns the minimum number of colours required to conflict-free colour the given hypergraph.

Corollary 8.2. *given a hypergraph H, $\chi_{cf}(H)$ can be found in $\mathcal{O}(2^n 2^m)$ time.*

Corollary 8.3. *The* Max-CFC *problem on hypergraphs induced by neighbourhoods of graphs can be solved in $\mathcal{O}(4^n)$ time.*

8.6 Exact Algorithm for Unique maximum Colouring

We now give an exact algorithm for solving Max-UMC on hypergraphs. It can be seen that the dynamic algorithm that we gave in Section 8.5, with minor changes, can be used to solve this problem.

Lemma 8.15. *There exists an exact algorithm to solve* Max-UMC *with running time $\mathcal{O}(3^n.2^m)$.*

Proof. Consider the vertex partitions given by the colouring. Observe that these partitions have the following property.

$$\forall F \in \mathcal{F}, \exists i \in [(H)] \text{ such that } |F \cap U_i| = 1 \text{ and } \forall j > i, F \cap U_j = \emptyset$$

This is similar to partitions given by a conflict-free colouring except for the last part. Let $\mathcal{U}$ be the function that takes a tuple $(X \subseteq U, \mathcal{E} \subseteq \mathcal{F})$ and maps it to the minimum number of colours required for unique-maximum colouring $(X, \mathcal{E}|_X)$. We give a recurrence very similar to that in the previous section.

$$\mathcal{U}(X, \mathcal{E}) = \begin{cases} \min_{X' \subseteq X, \forall E \in \mathcal{E}, |E \cap X'| = 0 \vee |E \cap X'| = 1} \{1 + \mathcal{U}(X \setminus X', \mathcal{E} \setminus \mathcal{E}')\}, & \text{if } X \neq \phi \\ 0, & \text{if } X = \phi \end{cases} \tag{8.3}$$

The correctness of this recurrence can be seen in a similar way. Assume X' is the maximum colour class in $\mathcal{U}$. Then X' uniquely colours all hyperedges that contains exactly one element from X'. The remaining hyperedges can be optimally coloured by $\mathcal{U}(X \setminus X', \mathcal{E} \setminus \mathcal{E}')$. Since we are considering all subsets of X, we get an optimal solution. □

8.7 Chapter Summary

We studied the Max-CFC and the Max-UMC problems and gave exact algorithms for the two problems. We also looked at p-CFC and p-UMC and gave an FPT algorithm that runs in time $2^{\mathcal{O}(k \log\log k + k \log r)} \cdot N^{\mathcal{O}(1)}$. Here, k is the number of hyperedges that are r-conflict-free coloured, and N is the size of the input instance. It would be interesting to show lower bounds for FPT algorithms for p-CFC and p-UMC. We also obtain an exponential vertex kernel for the problem, and it is open whether a polynomial kernel for the problem exists or not.

Chapter 9

Subexponential algorithms for rectilinear Steiner tree and arborescence problems

9.1 Introduction

In the Steiner Tree problem we are given as input a connected graph G, a non-negative weight function $w : E(G) \to \{1, 2, \ldots, W\}$, and a set of terminal vertices $T \subseteq V(G)$. The task is to find a minimum-weight connected subgraph of G, which is a tree, containing all terminal nodes T. Steiner Tree is one of the central and best-studied problems in Computer Science, we refer to the books of Hwang, Richards, and Winter [Hwang 1992] and Prömel and Steger [Prömel 2002] for thorough introductions to the problem.

In this chapter, we give the first *subexponential* algorithm for an important geometric variant of Steiner Tree, namely Rectilinear Steiner Tree. Here, for a given set of terminal points T in the Euclidean plane with ℓ_1-norm, the goal is to construct a network, of minimum length, connecting all points in T. This variant of the problem is extremely well studied, see Chapter 3 of the recent book of Brazil and Zachariasen [Brazil 2015] for an extensive overview of various applications of Rectilinear Steiner Tree.

Alternatively, it is convenient to define Rectilinear Steiner Tree as the Steiner Tree problem on a special class of graphs called Hanan grids. Recall that, for two points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ in the Euclidean plane $\mathbb{R}^2$, the *rectilinear* (ℓ_1, *Manhattan* or *taxicab*) distance between p_1 and p_2 is $d_1(p_1, p_2) = |x_1 - x_2| + |y_1 - y_2|$.

Definition 9.1 (Hanan grid [Hanan 1966])**.** *Given a set T of n terminal points in the Euclidean plane $\mathbb{R}^2$, the* Hanan grid *G of T is defined as follows. The vertex set $V(G)$ of G is the set of intersection points obtained by drawing a horizontal line (line parallel to x-axis) and a vertical line (line parallel to y-axis) through each point of T. For every $u, v \in V(G)$, there is an edge between u and v in G, if and only if u and v are adjacent along a horizontal or vertical line; the weight of edge uv is the rectilinear distance between u and v. For a Hanan grid G we define a weight function $\mathsf{recdist}_G$ from the edge set $E(G)$ to $\mathbb{R}$ such that for an edge $uv \in E(G)$, $\mathsf{recdist}_G(uv) = d_1(u, v)$. If the graph G is clear from the context we drop the subscript from $\mathsf{recdist}_G$ and only use* recdist*.*

When G is the Hanan grid of a set T of n points, then $T \subseteq V(G)$, $|V(G)| \leq n^2$, and for every $u, v \in V(G)$, the weight of a shortest path between u and v is equal to $d_1(u, v)$. For an edge $uv \in E(G)$, we say that uv is a *horizontal (vertical) edge* if both points u and v are on the same horizontal (vertical) line. It was shown by Hanan [Hanan 1966] that the

Rectilinear Steiner Tree problem can be defined as the following variant of Steiner Tree.

Rectilinear Steiner Tree
Input: A set T of n terminal points, the Hanan grid G of T and $\mathsf{recdist}_G$.
Output: A minimum Steiner tree for T in G.

Previous work on Rectilinear Steiner Tree. Though the Rectilinear Steiner Tree problem is a very special case of the Steiner Tree problem, the decision version of the problem is known to be NP-complete [Garey 1977]. A detailed account of various algorithmic approaches applied to this problem can be found in books of Brazil and Zachariasen [Brazil 2015], and Hwang, Richards, and Winter [Hwang 1992]. In particular, several exact algorithms for this problem can be found in the literature. The classic algorithm of Dreyfus and Wagner [Dreyfus 1971], from 1971, solves Steiner Tree on general graphs in time $3^n \cdot \log W \cdot |V(G)|^{\mathcal{O}(1)}$, where W is the maximum edge weight in G. For Rectilinear Steiner Tree, an adaptation of Dreyfus-Wagner algorithm provides an algorithm of running time $\mathcal{O}(n^2 \cdot 3^n)$. The survey of Ganley [Ganley 1999] summarises the chain of improvements based on this approach, concluding with the $\mathcal{O}(n^2 \cdot 2.62^n)$-time algorithm of Ganley and Cohoon [Ganley 1997]. Thomborson et al. [Thomborson 1987] and Deneen et al. in [Deneen 1994] gave randomized algorithms with running time $2^{\mathcal{O}(\sqrt{n}\log n)}$ for the special case of Rectilinear Steiner Tree when the terminal points T are drawn from a uniform distribution on a rectangle.

It is also worth mentioning relevant parameterized algorithms for Steiner Tree on general graphs. Fuchs et al. [Fuchs 2006] provide an algorithm with running time $\mathcal{O}((2+\varepsilon)^n |V(G)|^{f(1/\varepsilon)} \log W)$. Björklund et al. [Björklund 2007] and Nederlof [Nederlof 2013] gave $2^n |V(G)|^{\mathcal{O}(1)} \cdot W$ time algorithms for Steiner Tree. Let us remark that, since the distances between adjacent vertices in Hanan grid can be exponential in n, the algorithms of Björklund et al. and of Nederlof do not outperform the Dreyfus-Wagner algorithm for the Rectilinear Steiner Tree problem. Interesting recent developments also concern Steiner Tree on planar graphs, and more generally, on graphs of bounded genus. While the existence of algorithms running in time subexponential in the number of terminals on these graph classes is still open, Pilipczuk et al. [Pilipczuk 2013, Pilipczuk 2014] showed that Steiner Tree can be solved in time subexponential in the size of the Steiner tree on graphs of bounded genus.

In spite of the long history of research on Rectilinear Steiner Tree and Steiner Tree, whether Rectilinear Steiner Tree can be solved in time subexponential in the number of terminals remained open. In this chapter, a description of the first such algorithm is given. The running time of our algorithm is $2^{\mathcal{O}(\sqrt{n}\log n)}$. Further, our techniques also yield the first subexponential algorithm for the related Rectilinear Steiner Arborescence problem.

Definition 9.2. *Let G be a graph, $T \subseteq V(G)$ a set of terminals, and $r \in T$ be a root vertex. A* Steiner arborescence of T in G *is a subtree $H \subseteq G$ rooted at r with the following properties:*

- *H contains all vertices of T, and*
- *For every vertex $t \in T \setminus \{r\}$, the unique path in H connecting r and t is also the shortest $r \cdots t$ path in G.*

Let us note that if H is a Steiner arborescence of T in G, then for every vertex $v \in V(H)$, the unique path connecting r and v in H is also a shortest $r \cdots v$ path in G. The Rectilinear Steiner Arborescence problem is defined as follows.

Rectilinear Steiner Arborescence
Input: A set T of n terminal points, the Hanan grid G of T, a root $r \in T$ and $\mathsf{recdist}_G$.
Output: A minimum length Steiner arborescence of T.

Rectilinear Steiner Arborescence was introduced by Nastansky, Selkow, and Stewart [Nastansky] in 1974. Interestingly, the complexity of the problem was open until 2005, when Shu and Su [Shi 2000] proved that the decision version of Rectilinear Steiner Arborescence is NP-complete. No subexponential algorithm for this problem was known prior to our work.

Our method. Most of the previous exact algorithms for Rectilinear Steiner Tree exploit Hwang's theorem [Hwang 1976], which describes the topology of so-called full rectilinear trees. Our approach here is entirely different. The main idea behind our algorithms is inspired by the work of Klein and Marx [Klein 2014], who obtained a subexponential algorithm for Subset Traveling Salesman Problem on planar graphs. The approach of Klein and Marx was based on the following two steps: (1) find a locally optimal solution such that its union with some optimal solution is of bounded treewidth, and (2) use the first step to guide a dynamic program. While our algorithm follows this general scheme, the implementations of both steps for our problems are entirely different from [Klein 2014].

We give a high level description of the algorithm for Rectilinear Steiner Tree. The algorithm for Rectilinear Steiner Arborescence is similar. In the first step we build in polynomial time a (possibly non-optimal) solution. To build a non-optimal Steiner tree $\widehat{S}$ of $T = \{t_1, \ldots, t_n\}$, we implement the following greedy strategy. We build $\widehat{S}$ starting from vertex t_1 and gradually connect new terminals to the tree. When we connect terminal t_{i+1} to tree S_i spanning the first i terminals, we select a shortest monotone (containing at most one "bend") path from t_{i+1} to S_i in the Hanan grid. If there are two such paths, we select one of them according to the structure of S_i. The constructed tree $\widehat{S}$ can be seen as a "shortest path" rectilinear Steiner tree. The property of $\widehat{S}$ which is crucial for the algorithm is that there is an optimal Steiner tree S_{opt} such that graph $S = \widehat{S} \cup S_{\mathsf{opt}}$ is of treewidth $\mathcal{O}(\sqrt{n})$.

For the second step we have $\widehat{S}$ at hand and know that there exists a subgraph S of G of treewidth $\mathcal{O}(\sqrt{n})$, which contains an optimal Steiner tree S_{opt} and $\widehat{S}$. Of course, if the subgraph S was given to us, then finding S_{opt} in S could be done by a standard dynamic programming on graphs of bounded treewidth. However, we only know that such a subgraph S exists, albeit with the extra information that $\widehat{S} \cup S_{\mathsf{opt}} \subseteq S$. It turns out that this is sufficient in order to mimic the dynamic programming algorithm for bounded treewidth, to solve Rectilinear Steiner Tree in time $2^{\mathcal{O}(\sqrt{n}\log n)}$.

Let us recall the dynamic programming algorithm for Steiner Tree on a rooted tree decomposition $\mathcal{T} = (\mathbb{T}, \mathcal{X} = \{X_t\}_{t \in V(\mathbb{T})})$ of the input graph, see e.g. [Cygan 2015, Theorem 7.8]. For each node $t \in V(\mathbb{T})$, let V_t be the union of vertices contained in all bags corresponding to nodes of the subtree of $\mathbb{T}$ rooted at t and let S_t be the subgraph induced by V_t. Then, in the dynamic programming algorithm, for each t we store a set of states, capturing the interaction between a minimal Steiner tree and subgraph S_t; in particular the weight of a tree and the information about its connected components in S_t. It is

possible to ensure that all the information carried out in each state is "locally" defined, i.e., the information can be encoded by the elements of the bag X_t only. Therefore, at the root node, there is a state that corresponds to an optimal Steiner tree.

In our algorithm, we define *types*, which are analogous to the states stored at a node of a tree decomposition. A type stores all the information of its corresponding state. Since we do not know the tree decomposition $\mathcal{T}$, a type stores more "local" information, to take care of the lack of definite information about S. We guess some structural information about the virtual tree decomposition $\mathcal{T}$ of S. For example, we guess the height h of the rooted tree $\mathbb{T}$. In a rooted tree decomposition, the level of a node t is defined by the height of the subtree rooted at t. In our algorithm, we generate types over h levels. The intuition is that, for a node $t \in \mathbb{T}$ of level h', for each state, of t, that was required for the dynamic programming over $\mathcal{T}$, there is an equivalent type generated in level h' of our algorithm. This implies that, at level h, there is a type equivalent to a state that corresponds to an optimal Steiner tree in S. In fact, we show that any Steiner tree corresponds to exactly one type $\widehat{D}$. During the iterative generation of types, the type $\widehat{D}$ may be generated many times. One such generation corresponds to an optimal solution. So, the final step of the algorithm involves investigating all the occurrences of type $\widehat{D}$ in the iterative generation, and finding the weight of a minimum Steiner tree. As in dynamic programming, a backtracking step will enable us to retrieve a minimum Steiner tree of S, and therefore of G. This work is based on [Fomin 2016].

9.2 Preliminaries

For a set V, and two partitions $\mathcal{P}_1$ and $\mathcal{P}_2$ of V, the join operation results in the partition $\mathcal{P}$, that is the most refined partition of V such that each part of $\mathcal{P}_1$ and $\mathcal{P}_2$ is contained in a single part of $\mathcal{P}$. The resultant partition $\mathcal{P}$ is often denoted as $\mathcal{P}_1 \sqcup \mathcal{P}_2$. Given a part B of $\mathcal{P}$, by $\mathcal{P} \setminus B$ denotes removing the part B from $\mathcal{P}$.

Given a weight function $w : E(G) \to \mathbb{R}$, for a subgraph H of G, we use $w(H)$ to denote the number $\sum_{e \in E(H)} w(e)$. Furthermore, for two vertices s and t in $V(G)$, by the term *shortest path* between s and t we mean the shortest path with respect to the weight function w. Given two subgraphs G_1, G_2 of G, a shortest path between G_1 and G_2 is a path P between a vertex $u \in V(G_1)$ and a vertex $v \in V(G_2)$ such that, among the shortest paths for each possible pair $\{u' \in V(G_1), v' \in V(G_2)\}$, P has minimum length.

Treewidth. The concept of a tree decomposition and treewidth of a graph was defined in Section 1.2. In this chapter, we require a tree decomposition with more properties. A tree decomposition $(\mathbb{T}, \mathcal{X})$ is called a *nice tree decomposition* if $\mathbb{T}$ is a tree rooted at some node r where $X_r = \emptyset$, each node of $\mathbb{T}$ has at most two children, and each node is of one of the following kinds:

1. **Introduce node**: a node t that has only one child t', where $X_t \supset X_{t'}$ and $|X_t| = |X_{t'}| + 1$.
2. **Introduce edge node**: a node t labelled with an edge uv, with only one child t' such that $\{u, v\} \subseteq X_{t'} = X_t$. This bag is said to introduce uv.
3. **Forget vertex node**: a node t that has only one child t', where $X_t \subset X_{t'}$ and $|X_t| = |X_{t'}| - 1$.

4. **Join node**: a node t with two children t_1 and t_2, such that $X_t = X_{t_1} = X_{t_2}$.
5. **Leaf node**: a node t that is a leaf of $\mathbb{T}$, and $X_t = \emptyset$.

Additionally, we require that every edge is introduced exactly once. One can show that a tree decomposition of width t can be transformed into a nice tree decomposition of the same width t and with $\mathcal{O}(t|V(G)|)$ nodes, see e.g. [Cygan 2015].

9.2.1 Planar graph embeddings and minors

A graph is *planar* if it can be embedded in the plane. That is, it can be drawn on the plane in such a way that its edges intersect only at their endpoints. Formally, a planar embedding Π of a graph G consists of an injective mapping $\Pi : V(G) \to \mathbb{R}^2$ and a mapping Π of edges $uv \in E(G)$ to simple curves in $\mathbb{R}^2$ that join $\Pi(u)$ and $\Pi(v)$. The mapping ensures that for $e, f \in E(G)$, $\Pi(e) \cap \Pi(f)$ contains only the images of common end vertices. Also, for $e \in E(G)$ and $v \in V(G)$, $\Pi(v)$ is not an internal point of $\Pi(e)$. Now we define the notion of a minor of a graph G.

Definition 9.3. *A graph H is a minor of a graph G, denoted as $H \leq_m G$, if it can be obtained from a subgraph of G by a sequence of edge contractions.*

Notice that this implies that H can be obtained from G by a sequence of vertex deletions, followed by a sequence of edge deletions and finally a sequence of edge contractions.

We will need the following folklore observation.

Observation 9.1. *Suppose G, H are connected graphs such that H is a minor of G. Then H can be obtained from G only by edge deletions and contractions.*

Proof. If H is a minor of G, then there is a sequence of minor operations such that all vertex deletions are performed first and then all edge operations are performed. Let σ be such a sequence for our graphs G and H. Let V' be the vertex set of G deleted by σ. Since H is connected, The graph $G - V'$ is also connected. We show that there is a sequence σ' such that the vertex deletion operations do not disconnect the graph.

We look at the following graph G'. The vertex set of G' contains a special vertex v_{sp} and all vertices of V'. The edge set contains all edges with both end points in V'. Also, suppose for a vertex $v \in V'$, there is at least one vertex $u \in V(G) \setminus V'$ such that $uv \in E(G)$. Then $E(G')$ contains the edge (v_{sp}, v). Since G is connected, so is G'. Let $T_{G'}$ be a Breadth-first Search tree of G', rooted at v_{sp}. We reorder the vertex deletions of σ by deleting vertices with maximum distance from v_{sp}, in $T_{G'}$, to the neighbours of v_{sp}. The new sequence of minor operations has the property that vertex deletions do not disconnect the graph.

Now, a vertex deletion is equivalent to the contraction of an arbitrary incident edge, followed by the deletion of the remaining incident edges. Thus, we have exhibited a sequence of only edge operations to derive H from G. □

We will also be using the notion of a minor model.

Definition 9.4. *Let G and H be two connected graphs, and $H \leq_m G$. A* minor model*, or simply a* model*, of a graph H is a collection of pairwise disjoint vertex subsets $\mathcal{P}(H) = \{C_v \subseteq V(G) \mid v \in V(H)\}$ such that,*

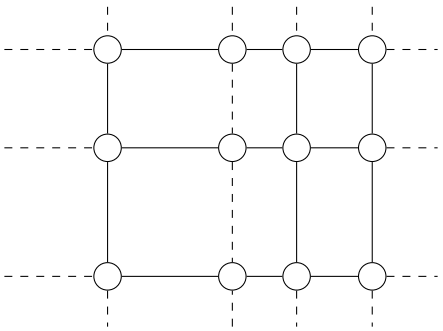


Figure 9.1: The solid edges define a subgrid of a grid.

(a) $V(G) = \biguplus_{v \in V(H)} C_v$,
(b) for each $v \in V(H)$, $G[C_v]$ *is connected, and*
(c) for any $uv \in E(H)$, *there exists* $w \in C_u$ *and* $w' \in C_v$ *such that* $ww' \in E(G)$.

Remark 9.1. *It is important to point out that in general the definition of a minor model does not demand that the vertex sets in* $\mathcal{P}(H) = \{C_v \subseteq V(G) \mid v \in V(H)\}$ *form a partition of* $V(G)$. *However, when both* G *and* H *are connected one can easily show that even this extra property can be assumed.*

Grids and subgrids play an important role for the results of this chapter. For a subset $W \subseteq [n]$, by $\max W$ ($\min W$) we denote the maximum (minimum) element of W.

Definition 9.5. *Let* n, m *be two positive integers. An* $n \times m$ *grid is a graph* G *such that* $V(G) = \{v_{i,j} \mid i \in [n], j \in [m]\}$ *and* $E(G) = \{v_{ij}v_{i'j'} \mid |i - i'| + |j - j'| = 1\}$. *For any* $i \in [n]$, *we call* $\{v_{i1}, \ldots, v_{im}\}$ *to be the* i*-th row of the grid* G *and for any* $j \in [m]$, *we call* $\{v_{1j}, \ldots, v_{nj}\}$ *to be the* j*-th a column of the grid* G. *The vertices in the first row,* n*-th row, the first column and* m*-th columns are called the* boundary vertices *of the grid. The vertices that are not boundary vertices are called* internal vertices.

The graph H *is called a subgrid of* G, *if there exist subsets* $R \subseteq [n], C \subseteq [m]$ *such that* $V(H) = \{v_{ij} \in V(G) \ : \ (\min R \leq i \leq \max R) \wedge (\min C \leq j \leq \max C) \wedge (i \in R \vee j \in C)\}$ *and* $E(H) = \{v_{ij}v_{i'j'} \in E(G) \ : \ v_{ij}, v_{i'j'} \in V(H) \wedge (i = i' \in R \vee j = j' \in C)\}$. *The set of vertices* $\{v_{ij} \in V(H) \ : \ i \notin \{\min R, \max R\} \vee j \notin \{\min C, \max C\}\}$ *are called the internal vertices of* H. *The set of vertices* $\{v_{ij} \in V(H) \ : \ i \in R \wedge j \in C\}$ *are called cross vertices. Finally, the set of vertices* $\{v_{ij} \in V(H) \ : \ i \notin R \vee j \notin C\}$ *are called subdivision vertices of* H. *(See Figure 9.1).*

Given a planar graph G with an embedding Π, we call the vertices of the outer face boundary vertices, and all other vertices internal vertices.

Definition 9.6. *Let* G *be a planar graph with a planar embedding* Π, *and* C *be a simple cycle of* G. *Let* p_∞ *be a point in the outer face of* G *in the embedding* Π. *Then removal of* C *from* $\mathbb{R}^2$ *divides the plane into two regions. The region that does not contain the point* p_∞ *is called the* internal region *of* C, *and the region containing* p_∞ *is called the* outer/external region *of* C. *A vertex in* $V(G)$ *is called internal if it lies in the internal region of* C, *and external if it lies in the external region of* C. *An edge in* $E(G)$ *is called an external edge if there is at least one point on its curve that lies in the external region. It*

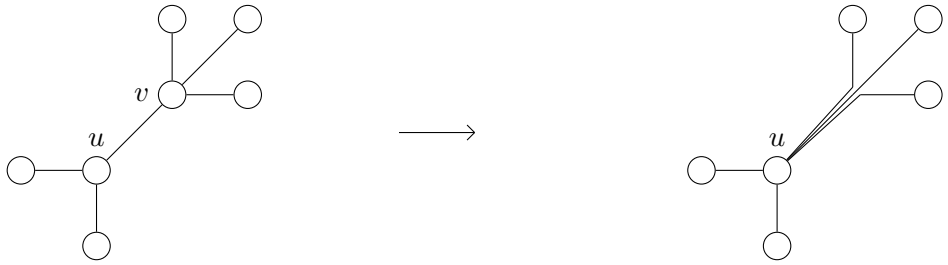


Figure 9.2: Derived embedding: the edge uv is contracted to the vertex u.

is called an internal edge if there is at least one point on its curve that lies in the internal region.

By definition of a planar embedding, an edge of $E(G)$ can be exactly one of the three kinds: an edge of $E(C)$, and external edge with respect to C or an internal edge with respect to C. Similarly a vertex can be exactly one of the three kinds: a vertex of C, an external vertex or an internal vertex.

Observation 9.2. *Let G be a planar graph with a planar embedding Π in $\mathbb{R}^2$. Let p_∞ be a point in the outer face of Π. Let H be a minor of G, and $\mathcal{P}(H) = \{C_v | v \in V(H)\}$ be a minor model of H. Then H is a planar graph. Furthermore, a planar embedding Π' of H can be obtained from Π that satisfies the following properties:*

- *Every vertex $v \in H$ is positioned in the place of a vertex in C_v.*
- *The point p_∞ is on the outer face of Π'.*

We call such a planar embedding Π' the embedding derived *from* Π.

Proof. It is well known that a minor of a planar graph is also a planar graph. We modify Π to obtain Π', in the following way:

1. A deletion operation changes the embedding of the current graph by simply deleting the element concerned. No other vertex or edge changes position.
2. Suppose the edge uv is to be contracted. Let the degree of v be d. we first make $d-1$ parallel copies of uv. We forget the vertex v and reroute the edges incident with v, other than uv using the original edges and one of the $d-1$ copies of uv (See Figure 9.2). The new embedding is a planar embedding of the new graph.

This ensures the two properties above. □

On the proof of the first step of the algorithm, we will use the following auxiliary lemma.

Lemma 9.1. *Let G and H be two connected planar graphs such that $H \leq_m G$ and let $\mathcal{P}(H) = \{C_v | v \in V(H)\}$ be a minor model of H in G. Let Π' be an embedding of H derived from a planar embedding Π of G. Suppose that H contains an induced subgraph H' isomorphic to a 3×3 grid. Let C' be the cycle formed by boundary vertices of H' and let v be the vertex of H' in the internal region of C'. Then there is a simple cycle C in G, such that:*

(1) $V(C) \subseteq \bigcup_{u \in V(H'); u \neq v} C_u$.

(2) For each vertex $w \in G$ that is contained in the internal region of C in Π, there is a vertex $u \in H'$ with $w \in C_u$.

(3) All vertices of C_v are completely contained in the internal region of C in Π.

(4) There is a vertex $w \in C_v$ such that $d_G(w) \geq 3$.

Proof. Consider consecutive vertices u, w in C'. The edge uw corresponds to an edge $u'w' \in E(G)$, such that $u' \in V(C_u), w' \in V(C_w)$. We will call edges like $u'w'$ marked edges in G. Note that both u' and v' do not belong to C_v. Now, for a boundary vertex u of H', consider the connected graph C_u of G. There are at most two vertices, u_1 and u_2, that are incident with marked edges in $V(C_u)$. Since $G[C_u]$ is a connected graph, there is a shortest path $P_{u_1u_2}$ connecting u_1 and u_2 in $G[C_u]$. We call $P_{u_1v_2}$ as a marked path. Note that $V(P_{u_1u_2}) \cap C_v = \emptyset$. The union of the marked edges and marked paths forms the simple cycle C and the vertex set $V(C)$ is disjoint from C_v. This proves condition (1). In fact a vertex of C only belongs to C_u for a vertex $u \in C'$.

Now we show condition 2. Consider a vertex w that is contained in the internal region of C in Π. Since G and H are both connected graphs, by Definition 9.4, there is a vertex $u \in H$ such that $w \in C_u$. If $u \in H'$, then condition 2 holds. Suppose not. Then u belongs to the external region of C' in Π'. By Observation 9.2, u is positioned at a vertex $w' \in C_u$. This means that C_u has a vertex w' in the external region of C and a vertex w in the internal region of C. Since $u \notin H'$, $C_u \cap C = \emptyset$. However, C_u is a connected subgraph of G and cannot be embedded without crossing with the cycle C. This is a contradiction to the fact that G is a planar graph. Hence, condition 2 must hold.

Next, we show that for the internal vertex $v \in H$, all vertices of C_v are completely contained in the internal region of C. From the definition of the derived embedding Π' from Π, as described in Observation 9.2, the point p_∞ is a point in the outer face of both embeddings. The internal vertex $v \in H$ is contained in the internal region of C'. Then, from the definition of Π' derived from Π, v is placed in the position of a vertex $u \in C_v$. By construction of C, it is disjoint from the vertices of C_v. Since C_v is connected, to maintain planarity, it follows that C_v is in the internal region of C.

Lastly, we show that for the internal vertex $v \in H$, there is a vertex $w \in C_v$ that has at least three neighbours in G. Notice that the induced subgraph G' of G, formed by the vertices of C and the internal region of C, also has H' as a minor. For a vertex $u \in H'$, let D_u denote the restriction of C_u in G'. Since, for the internal vertex $v \in H'$, C_v is completely contained in the internal region of C, $D_v = C_v$. There are four neighbours of $v \in H'$ in H'. For a neighbour u of v, let e be an edge between D_u and C_v. We call the endpoint of e, in C_v, a marked vertex. There are at most four marked vertices in C_v.

Suppose there are at most two marked vertices. This means that at least one marked vertex, say w, has at least two neighbours outside C_v. Since C_v is connected, and there are at least two marked vertices in C_v, the degree of w must be at least three in G.

Otherwise, there are at least three marked vertices $x_1, x_2, x_3 \in C_v$. Let P_{12} be a shortest path in C_v, between x_1 and x_2. Among the vertices of P_{12}, let w be the closest to x_3, and let the shortest path between w and x_3 be Q. If $w = x_3$, then w must be internal in P_{12} and thus has at least two neighbours in C_v. Since w is also a marked vertex, it has a third neighbour outside C_v, thereby making its degree in G at least three. Otherwise, suppose w is an internal vertex of P_{12}. Then, the two neighbours in P_{12} and a neighbour in Q

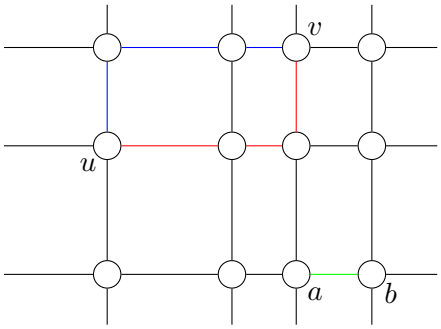


Figure 9.3: The red and blue paths are the two monotone $u \cdots v$ paths, and the green path is the single monotone $a \cdots b$ path.

make the degree of w at least three in G. Otherwise, w is one of x_1 or x_2. This means that w has a neighbour in P_{12}, x_3 as a neighbour, and a neighbour outside C_v. Thus, w has degree at least three in G. This completes the proof. □

Finally, our proof will be using the following version of the fundamental result of Robertson et al. [Robertson 1994].

Proposition 9.1 ([Gu 2012])**.** *Let t be a nonnegative integer. Then every planar graph G of treewidth at least $9t/2$ contains a $t \times t$ grid as a minor.*

9.2.2 Properties of shortest paths in the Hanan grid

Let G be the Hanan grid of a set of n points P. For a subgraph H of G, and $v \in V(H)$, we say that v is a *bend vertex* if there exists at least one horizontal edge and at least one vertical edge from $E(H)$ incident with v. A path $R = u_1 \cdots u_\ell$, between u_1 and u_ℓ in G, is called a *monotone path* if there exists $i \in [\ell]$ such that the points $u_1, \ldots, u_i$ belong to a horizontal line and $u_i, \ldots, u_\ell$ belong to a vertical line or vice-versa. In other words, all the horizontal edges as well as all the vertical edges in R are contiguous.

The following observation contains some simple facts about monotone paths (see Figure 9.3).

Observation 9.3. *Let u and v be two vertices of a Hanan grid G. Then,*

- *(a) There is at least one and at most 2 monotone $u \cdots v$ paths,*
- *(b) If the x-coordinates of u and v are equal, then there is only one monotone $u \cdots v$ path and all the edges in this path are vertical. Similarly, if the y-coordinates of u and v matches, the unique monotone $u \cdots v$ path consists of horizontal edges only.*
- *(c) If there are two monotone paths between u and v, then one path has a horizontal edge incident with u while the other path has a vertical edge incident with u*

Definition 9.7. *Suppose we are given a Hanan grid G of a set of terminals T and two vertices $u, v \in V(G)$. Let $x_1 = \min\{u_x, v_x\}$, $x_2 = \max\{u_x, v_x\}$, $y_1 = \min\{u_y, v_y\}$, and $y_2 = \max\{u_y, v_y\}$. Let $V' = \{w \in V(G) | w_x \in [x_1, x_2], w_y \in [y_1, y_2]\}$. Then $G' = G[V']$, the subgraph of G induced by V', is called a grid defined by the two vertices u, v as its diagonal points.*

Observation 9.4. *Given a Hanan grid G, a shortest path between any two vertices u, v has the property that the sequence of the x-coordinates of the vertices of the path is a monotone sequence, and the sequence of their y coordinates is also a monotone sequence.*

Observation 9.5. *Given a grid G, all shortest paths between two vertices u, v are contained in the grid $G' \leq_s G$ that is defined by u, v as its diagonal points. In fact, any path, with the property that the sequence of the x-coordinates of the vertices of the path is a monotone sequence and the sequence of their y coordinates is also a monotone sequence, and which is fully contained inside G', is a shortest path between u and v.*

9.3 Subexponential algorithm for Rectilinear Steiner Tree

In this section, we give a subexponential algorithm for Rectilinear Steiner Tree. Let T be an input set of terminals (points in $\mathbb{R}^2$), $|T| = n$, and G be the Hanan grid of T. Furthermore, let $\mathsf{recdist}_G$ denote the weight function on the edge set $E(G)$. For brevity we will use $\mathsf{recdist}$ for $\mathsf{recdist}_G$. The described algorithm is based on a dynamic programming over vertex subsets of size $\mathcal{O}(\sqrt{n})$ of G. To reach the stage where we can apply the dynamic programming algorithm, we do as follows. First, we define a rectilinear Steiner tree, called *shortest path RST*, and describe some of its properties. Next, we show that for a shortest path RST $\widehat{S}$, there is an optimal Steiner tree S_{opt} such that $\widehat{S} \cup S_{\mathsf{opt}}$ has bounded treewidth. Finally, keeping a hypothetical tree decomposition of $\widehat{S} \cup S_{\mathsf{opt}}$ in mind, we design a dynamic programming algorithm to obtain the size of a minimum rectilinear Steiner tree of G.

9.3.1 Shortest Path RST and its properties

In this part, we define a shortest path RST for a set $T = \{t_1, \ldots, t_n\}$ of input terminals and prove some useful properties of such a Steiner tree. We define a *shortest path RST* as follows.

Let G be the Hanan grid of T. We define a shortest path RST $\widehat{S}$ through the following constructive greedy process. Initially, we set S_1 to the graph $(\{t_1\}, \emptyset)$, which is a rectilinear Steiner tree of $\{t_1\}$. In the i^{th} step, we compute a rectilinear Steiner tree S_{i+1} of $\{t_1, \ldots t_{i+1}\}$ from S_i as follows. If $t_{i+1} \in V(S_i)$, then we set $S_{i+1} = S_i$. Otherwise, let v be a vertex in S_i such that $\mathsf{recdist}(v, t_{i+1}) = \min_{u \in V(S_i)} \mathsf{recdist}(u, t_{i+1})$. If there is only one monotone $t \cdots v$ path, then let Q be this path. Otherwise, there are two monotone $t \cdots v$ paths, such that one path has a horizontal edge incident with v and the other has a vertical edge incident with v. If there is a horizontal edge in S_i which is incident with v, then we choose Q to be the monotone $t \cdots v$ path such that the edge in Q incident with v is a horizontal edge. Otherwise we choose Q to be the monotone $t \cdots v$ path such that the edge in Q incident with v is a vertical edge. Then we construct S_{i+1} by adding the chosen monotone path Q to S_i. After $n-1$ iterations, we construct a tree $\widehat{S} = S_n$ of G, which is a Steiner tree of T. This is our shortest path RST.

It easy to see that one can construct a shortest path RST in polynomial time.

Lemma 9.2. *Given a set T of terminal points and the Hanan grid G of T, a shortest path RST $\widehat{S}$ of T can be constructed in polynomial time.*

Proof. Consider the procedure used to define a shortest path RST. The procedure involves $|T|$ steps. In each step, a shortest path, between two vertices of G, is found out. Since the shortest path subroutine can be executed in polynomial time, and there are polynomially many calls to this subroutine, the construction of $\widehat{S}$ requires polynomial time. □

Next, we give an upper bound on the number of bend vertices in a shortest path RST.

Lemma 9.3. *The number of bend vertices in $\widehat{S}$ is at most n.*

Proof. We prove the assertion by induction on the number of iterations to construct the solution $\widehat{S}$. Towards this, using induction on i, we prove that the number of bend vertices in S_i is at most i. In the base case, S_1 is a single vertex with no bend vertices. Suppose that the statement holds for S_{i-1}. If t_i is already contained in S_{i-1}, then $S_i = S_{i-1}$. By induction, the number of bend vertices in S_{i-1} is at most $i-1$, and therefore the number of bend vertices in S_i is at most i. Otherwise, we find a vertex $v \in V(\widehat{S}_{i-1})$ such that a shortest path Q between t_i and $\widehat{S}_{i-1}$ ends with v.

Since Q is a shortest path between t_i and $\widehat{S}_{i-1}$, the set of internal vertices of Q is disjoint from $V(\widehat{S}_{i-1})$. By induction hypothesis, S_{i-1} had at most $i-1$ bend vertices. The number of bend vertices in Q is at most 1. If v is already a bend vertex in S_{i-1}, then the number of bend vertices in S_i is at most $i-1+1=i$. By the way we define shortest path RST, if v is not a bend vertex in S_{i-1}, it is also not bend in S_i. Therefore, in this case, the number of bend vertices in S_i is also at most i. This concludes the proof. □

9.3.2 Supergraph of an optimal RST with bounded treewidth

We view the Hanan grid G as a planar graph, and use this viewpoint to obtain the required upper bound on the treewidth of a subgraph of G. In particular, given a shortest path RST $\widehat{S}$, we show the existence of an optimal Steiner tree S_{opt} such that the treewidth of $\widehat{S} \cup S_{\mathsf{opt}}$ is sublinear in the number of terminal points T. First, we show that there is an optimal Steiner tree in G that has a bounded number of bends.

Lemma 9.4. *Let T be a set of n points in $\mathbb{R}^2$ and G be the Hanan grid of T. Then there is an optimal rectilinear Steiner tree of T, such that the number of bend vertices in the rectilinear Steiner tree is at most $3n$.*

Proof. We prove the lemma using induction on $n = |T|$. The base case is when $n = 1$. Since the tree $(T, \emptyset)$ is an optimal Steiner tree when $|T| = 1$, the number of bend vertices in the tree $(T, \emptyset)$ is zero. Similarly, when $n = 2$, a monotone path between the two terminal vertices is an optimal Steiner tree. Therefore, the number of bends is one and the hypothesis is still true.

Consider the induction step where $n = |T| > 2$. Let S_{opt} be an optimal Steiner tree of T in G. Since S_{opt} is an optimal Steiner tree of $|T|$, there are at least 2 leaves and each leaf node is a terminal. Also, there is a pair $\{t_1, t_2\}$ of leaf terminals such that, in the $t_1 \cdots t_2$ path P of S_{opt}, there is at most one internal vertex with degree at least three in S_{opt}. This means that all other internal vertices in P are of degree exactly two in S_{opt}. If there are no internal vertices of degree at least three in P, this means that all the terminals in T are collinear and that S_{opt} is a path. Otherwise, let u be an internal vertex of P with degree

at least three in S_{opt}. Consider the sub-paths P_1 and P_2, which are $t_1 \cdots u$ and $t_2 \cdots u$ paths. Suppose there is a terminal t appearing as an internal vertex on P. Without loss of generality, we can assume that $t \in V(P_1)$ and t is the second terminal vertex in the path P_1 (first terminal vertex is t_1). Let us denote the $t_1 \cdots t$ sub-path of P_1 as P_3. By definition, all the internal vertices between t_1 and t are degree two non-terminal vertices. Let S_1 be the tree obtained by deleting $V(P_3) \setminus \{t\}$ from S_{opt}. Since S_{opt} is an optimal Steiner tree of T, S_1 is an optimal Steiner tree of $T \setminus \{t_1\}$ and P_3 is a minimum weight $t_1 \cdots t$ path. Since $|T \setminus \{t_1\}| = n-1$, by induction hypothesis, there is an optimal Steiner tree S' of $T \setminus \{t_1\}$ such that the number of bend vertices is at most $3(n-1)$. Let P_3' be a monotone $t_1 \cdots t$ path. Since $\mathsf{recdist}(S' \cup P_3') \leq \mathsf{recdist}(S_1 \cup P_3) = \mathsf{recdist}(S_{\mathsf{opt}})$, $S' \cup P_3'$ is an optimal Steiner tree of T. By the definition of a monotone path, the number of bend vertices in P_3' is at most 1. Thus, any bend vertex b in $S' \cup P_3'$ is a bend vertex in S', or a bend vertex in P_3', or $b = t$. This implies that the number of bend vertices in $S' \cup P_3'$ is at most $3(n-1) + 1 + 1 \leq 3n$.

The remaining case is that the $t_1 \cdots t_2$ path P has exactly one internal vertex u of degree at least three, while all other internal vertices are of degree two and are not terminals. Let S_1 be the tree obtained by deleting $V(P_1 \cup P_2) \setminus \{u\}$ from S_{opt}. Since S_{opt} is an optimal Steiner tree of T, S_1 is an optimal Steiner tree of $T \setminus \{t_1, t_2\} \cup \{u\}$. Again, the sub-paths P_1 and P_2 must be minimum weight $t_1 \cdots u$ and $t_2 \cdots u$ paths respectively. Since $|T \setminus \{t_1, t_2\} \cup \{u\}| = n-1$, by induction hypothesis, there is an optimal Steiner tree S' of $T \setminus \{t_1, t_2\} \cup \{u\}$ such that the number of bend vertices is at most $3(n-1)$. By optimality of S_{opt}, for any minimum weight $t_1 \cdots u$ path P_1' and $t_2 \cdots u$ path P_2', $\mathsf{recdist}(S' \cup P_1' \cup P_2') = \mathsf{recdist}(S_{\mathsf{opt}})$. Thus, $S = S' \cup P_1' \cup P_2'$ is an optimal Steiner tree of T. We choose P_1' and P_2' to be monotone paths. Thus, the number of bend vertices in P_1' and P_2' is at most one each. This means that u could be a new bend vertex in S, and there could be at most two new bend vertices in the two monotone paths added. This brings the total number of newly introduced bend vertices to at most three. Thus, the total number of bend vertices in S is at most $3n$. This completes the proof. □

With respect to a shortest path RST $\widehat{S}$, of G, we prove the next Lemma. In particular, we show that the treewidth of $S = \widehat{S} \cup S_{\mathsf{opt}}$ is at most $41\sqrt{n}$. Here, S_{opt} is a carefully chosen optimal Steiner tree for T. In order to get the desired upper bound on the treewidth of S, we show that it does not contain $\mathcal{O}(\sqrt{n}) \times \mathcal{O}(\sqrt{n})$ grid as a minor. In fact, we prove that if there is a large grid, then we can find a "clean part of the grid" (subgrid not containing vertices of T and bend vertices of either $\widehat{S}$ or S_{opt}), and reroute some of the paths in either $\widehat{S}$ or S_{opt}. This, in turn, contradicts either the way $\widehat{S}$ is constructed or the optimality of S_{opt}.

Lemma 9.5. *Given a set T of n points and a shortest path RST $\widehat{S}$ of T, there is an optimal rectilinear Steiner tree S_{opt} of T with the property that the treewidth of $\widehat{S} \cup S_{\mathsf{opt}}$ is bounded by $41\sqrt{n}$.*

Proof. Among the optimal Steiner trees of T with the minimum number of bend vertices, we select a tree S_{opt} which has maximum edge intersection with $E(\widehat{S})$. From Lemma 9.4, it follows that the number of bend vertices in S_{opt} is at most $3n$.

Let $S = \widehat{S} \cup S_{\mathsf{opt}}$. Let $\widehat{B}$ and B_{opt} be the set of bend vertices in $\widehat{S}$ and S_{opt}, respectively. Let $U = T \cup \widehat{B} \cup B_{\mathsf{opt}}$ and $N = V(G) \setminus U$. Since $|T| = n$, $|\widehat{B}| \leq n$, and $|B_{\mathsf{opt}}| \leq 3n$, we know that $|U| \leq 5n$. Let Π_S be a planar embedding of S, obtained by deleting all

the edges and vertices not in S from the planar embedding Π of G. We show that the treewidth of S is at most $41\sqrt{n}$. We can assume that $n \geq 4$, as otherwise we can greedily find out the best rectilinear Steiner tree from the constant sized Hanan grid. For the sake of contradiction, assume that $\mathbf{tw}(S) > 41\sqrt{n}$. Then, by Proposition 9.1, there is a $9\sqrt{n} \times 9\sqrt{n}$ grid H appearing as a minor of S. Let $\mathcal{P}(H) = \{C_v | v \in V(H)\}$ be a minor model of H. Since H and G are connected graphs, $\mathcal{P}(H)$ is a partition of the vertex set $V(G)$. We identify a 3×3 subgrid H' of H by the following process. For any $v \in V(H)$, we mark the vertex v if $C_v \cap U \neq \emptyset$ (i.e, C_v contains a terminal or a bend vertex from $\widehat{S}$ or S_{opt}). Since $|U| \leq 5n$, the number of marked vertices in H is at most $5n$. Since H is a $9\sqrt{n} \times 9\sqrt{n}$ grid, there are at least $6n$ vertex disjoint 3×3 subgrids in H. This implies that there is a 3×3 subgrid H' in H such that each vertex of H' is unmarked. The fact that for $u \in V(H')$, $C_u \cap U = \emptyset$ implies the following observation.

Observation 9.6. *Let $u \in V(H')$ and $w \in C_u$.*

(i) $d_{\widehat{S}}(w), d_{S_{\mathsf{opt}}}(w) \in \{0, 2\}$. If for any $S_i \in \{\widehat{S}, S_{\mathsf{opt}}\}$, $d_{S_i}(w) = 2$, then the two edges in S_i incident with w are of same kind (either horizontal or vertical).

(ii) If one horizontal (vertical) edge incident with w is present in S, then the other horizontal (vertical) edge incident with w is also present in S. Hence $d_S(w) \in \{2, 4\}$.

Note that H' is a connected graph and is a minor of a connected graph S. Let $\Pi_{H'}$ be a planar embedding derived from Π_S. By Lemma 9.1, we know that there is a simple cycle C' in S with the following properties.

(i) $V(C') \subseteq \bigcup_{u \in V(H')} C_u$.

(ii) For each vertex $w \in G$ that is contained in the internal region of C' in Π, there is a vertex $u \in H'$ with $w \in C_u$. In particular, all the vertices of $V(S) \setminus \bigcup_{v \in V(H')} C_v$ (which includes U) are not in the internal region of C'.

(iii) For the internal vertex $v \in V(H')$, all the vertices in C_v are in the internal region of C'.

(iv) Finally, there is a vertex $w \in C_v$, in the internal region of C', such that $d_S(w) \geq 3$. By Observation 9.6, $d_S(w) = 4$.

That is, there is a cycle C' in the Hanan grid G such that $V(C') \subseteq V(S) \setminus U$, every point in the internal region of C' does not correspond to any vertex in U and there is a vertex w of degree 4 in S, which is in the internal region of C'. The following claim follows from Observation 9.6.

Claim 9.1. *Let $u, v \in V(G)$ be points which are either on the same horizontal line or on the same vertical line. If the line segment L connecting u and v does not intersect with the outer region of C', and there is an edge $v_1v_2 \in E(S)$ on the line L, then all the edges on the line segment L belong to $E(S)$.*

Let C' be a minimum-weight cycle satisfying properties $(i), (ii)$ and (iv) and w' be a vertex of degree 4 in the internal region of C'. Let $E_{w'}$ be the set of edges of S not in the outer region of C' and each edge either belongs to the horizontal line or vertical line containing w'.

Claim 9.2. *Graph $G' = C' \cup E_{w'}$ is a subgrid of G. Moreover, $V(G') \subseteq V(G) \setminus U$, $E(G') \subseteq E(S)$, and all the subdivision vertices in G' have degree exactly 2 in S.*

Proof. The definition of G' implies that $V(G') \subseteq V(G) \setminus U$ and $E(G') \subseteq E(S)$. Let L_1 and L_2 be the horizontal and vertical line passing through the point $w' = (w'_x, w'_y)$ respectively. We show that G' is indeed a subgrid of G. Let l, r be degree 4 vertices of S on the line L_1 such that $l_x < w'_x, r_x > w'_x$, and the distances $\mathsf{recdist}(w', l)$ and $\mathsf{recdist}(w', r)$ are minimized. Similarly, let a, b be degree 4 vertices of S on the line L_2 such that $a_y > w'_y, b_y < w'_y$ and the distances $\mathsf{recdist}(w', a)$ and $\mathsf{recdist}(w', b)$ are minimized. Let $R = \{a_y, w'_y, b_y\}$ and $C = \{l_x, w'_x, r_x\}$. Now we will show that the subgrid G'', of G, defined by R and C is same as G'. Let L'_1 be the line segment of L_1, between l and r. This line segment is not in the external region of C'. Similarly, the line segment L'_2 on L_2, between a and b, is not in the external region of C'.

Let C'' be the cycle formed by the boundary vertices of G''. We need to show that C'' is the same as C'. We first show that **(a)** there is no edge $uv \in E(S)$ such that uv is in the internal region of C'' and $uv \in E(S) \setminus E(G'')$. Suppose not. Among all such edges, let uv be an edge such that $\mathsf{recdist}(w', u)$ is minimized in the Hanan grid G. As uv does not belong to $E(G'')$, it does not lie on the line segments L'_1 and L'_2. Since uv is an internal edge of C'', $l_x < u_x < r_x$ and $b_y < u_y < a_y$. Notice that any shortest path, in G, between u and w' lies in the internal region of C''. In other words, the grid G_u defined by the u and w' lies in the internal region of C''. Any edge in G_u has shorter distance to w' than uv. Thus, since uv has minimum distance to w', if an edge, of G_u, does not belong to L'_1 or L'_2, then the edge cannot belong to $E(S)$. We show that uv is not in the external region of C'. Suppose uv is in the external region of C'. Since, w' is in the internal region of C', a shortest path from u to w' must cross into the internal region bounded by C'. As L'_1 and L'_2 are also not in the external region of C', there is an edge of $E(G_u) \setminus (L'_1 \cup L'_2)$, that belongs to C'. Therefore, there is an edge in G_u that belongs to S. This edge belongs to $E(S) \setminus E(G'')$, is in the internal region of C'', but is closer to w' than uv. This contradicts the choice of uv. Thus, uv is not in the external region of C'. Consider the line L passing through uv. As mentioned earlier, $L \notin \{L_1, L_2\}$. Then L hits the line L_i, where L_i is exactly one of L_1 or L_2, at a single point z. First, suppose $u \neq z$. Let the line segment L' of L connect u and z. As shown above, u either belongs to C', or is in the internal region of C'. Following from Observation 9.6, since uv is an edge of S, there is another edge ux incident with u and lying on the line segment L'. This edge is also in the internal region of C'' and does not belong to $E(G'')$. Consider the other endpoint x. This vertex has shorter distance to w' than u. This contradicts the fact that uv was the chosen edge.

Finally, if $u = z$, then u lies on the line L_i. The line segments of L'_1 and L'_2 are not in the external region of C'. Note that, $l_x < u_x < r_x$ and $b_y < u_y < a_y$. This means that the edge uv as well as the two edges of L_i, incident on u, belong to S and are not in the external region of C'. Hence, there are both horizontal and vertical edges in S which are incident with u. Since u is not an external vertex of C', the degree of u in S is 4 (by Observation 9.6). However, u is a vertex on L_i, $l_x < u_x < r_x$ and $b_y < u_y < a_y$. This contradicts the choice of one of l, r, a or b.

The condition (a) implies that the internal region of C'' does not have an edge of S. Since all the vertices $\{\ell, r, a, b, w'\}$ either belong to C' or to the internal region of C', the internal region of C'' is a subset of the internal region of C'. Also, all the edges in C'' are not in the outer region of C'. Since the degree of l, r, a and b in S is 4, by Claim 9.1, all the edges in C'' belong to $E(S)$. Since w' is in the internal region of C'', condition (iv) holds for C''. Also the vertices and edges of C'' either belong to C' or are in the internal region of C'. Thus conditions (i) and (ii) also hold. Then, by the minimality of C', $C'' = C'$.

Since the degree of w' in S is 4, by Claim 9.1, all the edges in $E_{w'}$ belong to $E(S)$.

Now, we need to show that all the subdivision vertices of G' have degree 2 in S. Suppose not. Let u be a subdivision vertex in G' such that degree of u in S is greater than 2. By Observation 9.6, degree of u in S is 4. This implies that there is an edge uv in the internal region of C' and $uv \notin E(G')$. This contradicts condition (a). We have shown that $G' = C' \cup E_{w'}$ is a subgrid, where all subdivision vertices are of degree 2 in S. □

The next claim provides us with the insight on how subpaths of $\widehat{S}$ and S_{opt} behave in G'.

Claim 9.3. *Let F_h and F_v be the sets of horizontal and vertical edges in $G' = C' \cup E_{w'}$ respectively. Then exactly one of the following conditions is true.*

1. *$F_h \subseteq E(\widehat{S})$ and $F_v \subseteq E(S_{\mathsf{opt}})$.*
2. *$F_v \subseteq E(\widehat{S})$ and $F_h \subseteq E(S_{\mathsf{opt}})$.*

Proof. Let $G_1 = G'[E(\widehat{S})]$ and $G_2 = G'[E(S_{\mathsf{opt}})]$. First, we show that each component of G_1 and G_2 is a path where all edges are of the same kind, i.e., either all are horizontal or all are vertical. Note that a component of G_1 or G_2, with only horizontal or only vertical edges, must be a path. For contradiction's sake, suppose there is a component with both horizontal and vertical edges. Without loss of generality, we assume that there is a component $C_1 \in G_1$ with both kinds of edges. This implies that there is a vertex $v \in V(C)$ such that v is incident with a vertical edge as well as a horizontal edge. However, by Observation 9.6(i) and the definition of G', such a vertex cannot be in G'. Therefore, each component of G_1 and G_2 is a path where all edges are of the same kind.

Next, we show that the edges of G_1 are either all horizontal or all vertical. For ease of notation, we will call a set of edges, which are all horizontal or all vertical, to be *parallel* edges. Let D be a component of G_1 and e an edge of $E(G_1) \setminus E(D)$. Also, the edges of D are of a different orientation than the edge e. That is, if all the edges of D are horizontal, then e is a vertical edge, and vice-versa. As shown above, all the edges of D are parallel to each other. Among all such pairs (D, e) we choose a pair (C_1, uv) that has the minimum distance between D and e in G'. As G' is connected, there is a path between C_1 and uv. Without loss of generality, assume that all edges of C_1 are horizontal, and that uv is a vertical edge. Assume that u and a vertex $w \in C_1$ are the vertices whose shortest path Q_{uw} in G' is a witness to the vertical edge uv and the component C_1 having the minimum distance between them. We first show that no edge of Q_{uw} belongs to $E(\widehat{S})$. Traversing from u along Q_{uw}, let e^* be the first edge of $E(\widehat{S})$ that is encountered. If e^* is a vertical edge, then (C_1, e^*) is a pair which satisfies the above description. However, the distance between C_1 and e^* is strictly smaller than the distance between C_1 and uv, which is a contradiction. On the other hand, suppose that e^* is a horizontal edge and that it belongs to the component $C_2 \neq C_1$ of G_1. Then (C_2, uv) is a closer pair than (C_1, uv). Thus, all edges of Q_{uw} belong to $E(S_{\mathsf{opt}})$ and not to $E(\widehat{S})$. Moreover, all the edges of Q_{uw} must belong to a single component C_2 of G_2. This implies that all the edges of Q_{uw} are parallel to each other. Let e_w be the edge of Q_{uw} that is incident with w. Since w has at least one horizontal edge of $E(C_1) \subseteq E(\widehat{S})$ and $e_w \notin E(\widehat{S})$, by Observation 9.6(i) with respect to w, e_w must be a vertical edge. Since, C_2 is a component of G_2, all edges of Q_{uw} must be vertical edges. Let e_u be the edge of Q_{uw} incident with u. Both $e_u \in G_2$ and $e \in G_1$ are vertical edges, where $e \in E(\widehat{S})$ while $e_u \notin E(\widehat{S})$. This is a contradiction

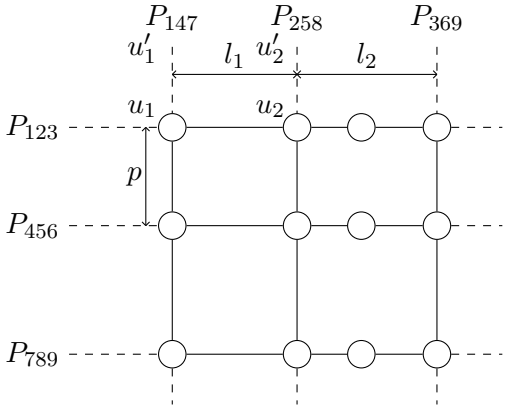


Figure 9.4: The subgrid G'.

to Observation 9.6(i) with respect to u. A similar argument shows that $E(G_2)$ is a set of parallel edges. This completes the proof of Claim. □

Note that G' is a 3×3 subgrid of G. Let G' be formed by horizontal paths $P_{123}, P_{456}, P_{789}$ and vertical paths $P_{147}, P_{258}, P_{369}$. Let $u_1, \ldots, u_9$ be the 9 vertices in G' such that the path P_{ijk}, where $i, j, k \in [9]$, contains the vertices u_i, u_j and u_k. Due to Claim 9.3, without loss of generality, we may assume that the horizontal paths belong to S_{opt}, and the vertical paths belong to $\widehat{S}$. For a path P_{ijk}, we use P_{ij} and P_{jk} to denote the sub-paths of P_{ijk} connecting u_i and u_j, and u_j and u_k respectively. Let the length of the sub-path P_{12} be l_1 and the length of the sub-path P_{23} be l_2. By the definition of G', the length of P_{45} is also l_1, and the length of P_{56} is l_2. (See Figure 9.4).

Suppose $l_1 + l_2 > 2p$. Then, we consider the graph S^* formed by deleting in S_{opt} the path P_{123}, and adding the two paths P_{14} and P_{36}. Since all the subdivision vertices of G' are of degree 2 in S, S^* is a Steiner tree of weight strictly less than the weight of S_{opt}. This contradicts the choice of S_{opt}. Hence, this is not possible.

Suppose $l_1 + l_2 \leq 2p$. Without loss of generality, let $l_1 \leq l_2$. Thus, $l_1 \leq p$. Consider the two paths P_{147} and P_{258}. They are vertical paths of $\widehat{S}$, such that all the vertices in these paths belong to $V(G) \setminus U$ (that is, non-terminals and non-bend vertices) and have degree 2 in $\widehat{S}$ (by Observation 9.6). This implies that all the edges in any path $R \in \{P_{147}, P_{258}\}$ are added in a single step while constructing $\widehat{S}$. Since both the paths are parallel, by Observation 9.4, if a path R_1 in $\widehat{S}$ has both P_{147} and P_{258} as sub-paths, then R cannot be a shortest path between its endpoints. Thus, by construction of $\widehat{S}$, both P_{147} and P_{258} could not have been added to $\widehat{S}$ in a single step of the construction. Also by construction, one of them is added to $\widehat{S}$ before the other. Without loss of generality, let P_{147} be added before P_{258}. Again by construction, a path, containing P_{258} as a sub-path, was added in the i^{th} step, to connect a terminal t to the already constructed $\widehat{S}_{i-1}$. Let R^* be the path added to S_{i-1}. By definition of G', this terminal t must lie outside the region formed by the subgrid G'. Since P_{258} was part of a shortest path between $\widehat{S}_{i-1}$ and t, by Observation 9.4, t must lie on a row strictly higher than or strictly lower than the rows in G'. Suppose that this terminal lies above P_{123}. By Observation 9.6, both the vertical edges incident on u_1 belong to $\widehat{S}$. Since u_1 and u_2 are of degree 2 in $\widehat{S}$, both the vertical edges incident with u_1 as well as u_2 are added, when the path containing P_{147} or P_{258} is added to construct $\widehat{S}$.

This implies that both the vertical edges, incident with u_1, are present in S_{i-1}. Let u_1u_1' be the vertical edge present in S_{i-1} and not in $E(P_{147})$. Let u_2u_2' be the vertical edge not present in P_{258}. Now, consider the path R_2, between u_2' and t, obtained by concatenating the horizontal path between u_1' and u_2', and the sub-path of R^* connecting u_2' and t. The length of the path R_2 is strictly less than that of R^* and $u_1' \in S_{i-1}$. This is a contradiction. The case when t lies below any row in G' is identical to the other case.

Thus, there is no such subgrid G' of G, such that G' is a subgraph of S. This implies that there is no $9\sqrt{n} \times 9\sqrt{n}$ grid as a minor in S. Due to Proposition 9.1, the treewidth of S must be at most $\frac{9}{2} \cdot 9\sqrt{n} = 41\sqrt{n}$. This completes the proof. □

9.3.3 Dynamic Programming Algorithm for Rectilinear Steiner Tree

In this section we utilize all the results proved in the previous sections and design our algorithm for Rectilinear Steiner Tree. By Lemma 9.5, we known that given a shortest path RST $\widehat{S}$, there exists an optimum Steiner tree S_{opt} such that the treewidth of $S = \widehat{S} \cup S_{\mathsf{opt}}$ is bounded by $41\sqrt{n}$. The idea of the algorithm is to *implicitly* do a dynamic programming over a tree decomposition of S, *even though we do not know what S is*, to compute an optimum Steiner tree for T.

Suppose we know the subgraph S of G, such that there is an optimum Steiner tree fully contained in S. Then we can do the well known algorithm for Steiner tree over the tree decomposition of S [Cygan 2015]. To give a proper intuition to our algorithm, we first recall the important step of the dynamic programming algorithm for Steiner tree over the tree decomposition of the graph S (see [Cygan 2015, Theorem 7.8] for more details). Let $(\mathbb{T}, \mathcal{X} = \{X_t\}_{t \in V(\mathbb{T})})$ be a nice tree decomposition of S, where $\mathbb{T}$ is a rooted tree. For a node t, let V_t be the union of all the bags present in the subtree of $\mathbb{T}$ rooted at t. For a node t, we define a graph $S_t = (V_t, E_t = \{e \in E(S) \ : \ e \text{ is introduced in the subtree rooted at } t\})$. The important step in the algorithm for Steiner Tree is to compute the following information: for each bag X_t, $X \subseteq X_t$ and a partition $\mathcal{P} = (P_1, \ldots, P_q)$ of X, the value $c[t, X, \mathcal{P}]$ is the minimum weight of a subgraph F of S_t with the following properties:

1. F has exactly q connected components $C_1, \ldots, C_q$ such that $\emptyset \neq P_i = X_t \cap V(C_i)$ for all $i \in [q]$. That is, $\mathcal{P}$ corresponds to connected components of F.
2. $X_t \cap V(F) = X$. That is, the vertices of $X_t \setminus X$ are untouched by F.
3. $T \cap V_t \subseteq V(F)$. That is, all the terminal vertices in S_t belong to F.

For our purpose, the second step is redundant but we still carry it out to give a proper analogy with the known algorithm for Steiner Tree over a graph of bounded treewidth we are referring to. Note that the number of parts in the partition $\mathcal{P}$ is q. Throughout this section, q will denote the number of parts of the partition in question.

It is known that computing the values $c[t, X, \mathcal{P}]$ for each tuple $(t, X, \mathcal{P})$, where $t \in V(\mathbb{T}), X \subseteq X_t$ and $\mathcal{P}$ is a partition of X, is enough to compute the value of an optimum Steiner tree of T in S. Also, this can be computed in time $\mathbf{tw}^{\mathcal{O}(\mathbf{tw})}|V(S)|^{\mathcal{O}(1)}$ (See Chapter 7 in the book [Cygan 2015]). In our case, we do not know the graph $S = \widehat{S} \cup S_{\mathsf{opt}}$ and a tree decomposition of S, but we know that the treewidth of S is at most $41\sqrt{n}$. This implies that the number of choices for bags in a tree decomposition of S is bounded by $n^{\mathcal{O}(\sqrt{n})}$. Consider the properties 1 and 2 mentioned above. They are *local* properties

of the witness subgraph F with respect to the bag X_t. However, the property 3 says that all the terminals in the subgraph S_t should be present in F. In fact, we can bound the *potential* sets $T \cap V(S_t)$, using the rectilinear Steiner tree $\widehat{S}$. Observe that any bag X_t is a separator of size $\mathcal{O}(\sqrt{n})$ for S and thus for $\widehat{S}$. This implies that *for every connected component C of $\widehat{S} - X_t$, either $T \cap V(C)$ is fully in V_t or no vertex in $T \cap V(C)$ belongs to V_t*. Since each vertex in G has degree at most 4 and X_t is a bag in a tree decomposition, the number of connected components in $\widehat{S} - X_t$ is at most $4|X_t| \leq 164(\sqrt{n}+1)$. As observed before, for any connected component C of $\widehat{S} - X_t$, either $T \cap V(C)$ is fully in V_t or no vertex in $T \cap V(C)$ belongs to V_t. This implies that the potential sets $T' \subseteq T$ such that $T' = (V_t \setminus X_t) \cap T$ is bounded by $2^{164(\sqrt{n}+1)}$ and we can enumerate them in sub-exponential time. Using this observation we could keep track of property 3 as well, even though we do not know the graph S and its tree decomposition.

Now, we move towards the formal explanation of our algorithm. We first define the notion of a *type*, which is the analogue of a tuple $(t, X, \mathcal{P})$ in the normal dynamic programming we explained above.

Definition 9.8. *A type is a tuple $(Y, Y' \subseteq Y, \mathcal{P}, T')$ such that the following holds.*

(i) Y is a subset of $V(G)$ of size at most $41\sqrt{n}+2$.
(ii) $\mathcal{P}$ is a partition of Y'.
(iii) There exists a set of components $C_1, \ldots, C_q$ in $\widehat{S} - Y$ such that $T' = T \cap (Y' \cup \bigcup_{i=1}^{q} V(C_i))$.

Informally, in a type $(Y, Y', \mathcal{P}, T')$, Y represents a potential bag Y of a node (say t) in a tree decomposition of S. The set Y' and partition $\mathcal{P}$ have the same meaning as that of $(t, Y', \mathcal{P})$ in the normal dynamic programming algorithm for Steiner Tree. The set T' is the set of terminals in the graph S_t. The next lemma gives an upper bound on the number of types.

Lemma 9.6. *There is a $2^{\mathcal{O}(\sqrt{n}\log n)}n^{\mathcal{O}(1)}$ time algorithm $\mathcal{B}$ enumerating all the types.*

Proof. We know that a type is a tuple $(Y, Y', \mathcal{P}, T')$ satisfying properties $(i) - (iii)$. Since $|V(G)| \leq n^2$, the number of choices for Y is $n^{\mathcal{O}(\sqrt{n})}$. The cardinality of Y is at most $41\sqrt{n}+2$, thus for a fixed Y, the number of choices for Y' is $2^{\mathcal{O}(\sqrt{n})}$ and the number of choices for the partition $\mathcal{P}$, of Y', is $n^{\mathcal{O}(\sqrt{n})}$. By definition of the Hanan grid G, each vertex in G has at most 4 neighbours. Thus, $\widehat{S} - Y$ has at most $4(41\sqrt{n}+2)$ components. Hence, on fixing Y, the choices for T' is at most $2^{\mathcal{O}(\sqrt{n})}$. Thus we get an upper bound of $2^{\mathcal{O}(\sqrt{n}\log n)}$ on the number of types. Furthermore, it can be enumerated in time $2^{\mathcal{O}(\sqrt{n}\log n)}n^{\mathcal{O}(1)}$. □

Our algorithm is a dynamic programming algorithm over the types of S. As motivated earlier, this algorithm essentially describes the ideas of the dynamic programming algorithm for Steiner Tree over a tree decomposition of an input graph. Let $N = 3(|V(G)| + |E(G)|)$. Our algorithm computes values $\mathcal{A}[i, D]$, where $i \in [N]$ and D is a type. We want the table $\mathcal{A}[,]$ to contain all the information that is necessary for the correctness of the dynamic programming algorithm for Steiner Tree over a tree decomposition of S. To motivate the definition of $\mathcal{A}[,]$, we assume a *hypothetical tree decomposition* $\mathcal{T} = (\mathbb{T}, \mathcal{X} = \{X_t\}_{t \in V(\mathbb{T})})$ of S. For ease of understanding, let it be a nice tree decomposition and let the tree be rooted at a node $r \in \mathbb{T}$. The level of a vertex $t \in \mathbb{T}$ is the height of the subtree of $\mathbb{T}$ rooted at t. The *height* of a node t is the number of

vertices in the longest downward path to a leaf from that node. Note that the level of any node of $\mathbb{T}$ is at most N. Suppose $t \in \mathbb{T}$ is a node at level i, and corresponds to the bag X_t. Let V_t be the union of bags present in the subtree rooted at t. Let the graph S_t be defined as $(V_t, \{e \mid e \text{ is introduced in a the subtree rooted at } t\})$. Let $T' = V_t \cap T$. Then, for any $X \subseteq X_t$, and a partition $\mathcal{P}$ of X having q parts, $\mathcal{A}[i, X_t, X\mathcal{P}, T')] = c[t, X, \mathcal{P}]$. As mentioned before, $c[t, X, \mathcal{P}]$ is the minimum weight of the subgraph F of S_t such that the following hold: (i) F has q connected components $C_1, \ldots, C_q$ such that $\emptyset \neq P_i = X_t \cap V(C_i)$, (ii) $X_t \cap V(F) = X$, and (iii) $T \cap V_t \subseteq V(F)$. For other pairs (i, D), we do not guarantee that the value of $\mathcal{A}[i, D]$ is meaningful. However, it is enough to maintain reasonable values for only the above subset of pairs (i, D). Of course, we do not know S and thus we do not know the tree decomposition $\mathcal{T}$. So, we store values in the table $\mathcal{A}[,]$ in such a way that given *any* nice tree decomposition of S, we have information pertaining to it. Thus, given a pair (i, D) where $D = (Y, Y' \subseteq Y, \mathcal{P}, T')$, we view Y as a bag of some hypothetical nice tree decomposition, $\mathcal{T}$, of S and assume that the level of the bag corresponding to Y in $\mathcal{T}$ is i. At a level i of this hypothetical nice tree decomposition, any bag is one of at most five kinds. We guess the relationship between a bag at level i and its children, which must be at level $i - 1$. For example, if our hypothetical node t corresponds to an introduce vertex bag X_t, then we pretend that we know X_t, the child node t', the bag $X_{t'}$, and the vertex v that is being introduced at node t. Thereafter, for a subset $X \subseteq X_t$, and a partition $\mathcal{P}$ of X, we try to compute $\mathcal{A}[i, (X_t, X, \mathcal{P}, T')]$ using that values of $\mathcal{A}$ calculated at step $i-1$ of the algorithm. The calculation ensures that $\mathcal{A}[i, (X_t, X, \mathcal{P}, T')] = c[t, X, \mathcal{P}]$. In what follows we give a formal definition of $\mathcal{A}[,]$.

We write a recurrence relation for $\mathcal{A}[i, D]$, where $i \in [N]$ and D is a type. We fix a terminal t^* in T

$$\mathcal{A}[1, D] = \begin{cases} 0 & \text{if } D = (\{t^*\}, \{t^*\}, \{\{t^*\}\}, \{t^*\}) \\ \infty & \text{otherwise} \end{cases} \tag{9.1}$$

To define $\mathcal{A}[i, D]$ for $i \in [N] \setminus \{1\}$ and a type $D = (Y, Y', \mathcal{P}, T')$, we first define many intermediate values and take the minimum over all such values.

We first try to *view* Y as an introduce node in some nice tree decomposition of S and having level i. This viewpoint results in the following recurrence. For all $v \in Y$,

$$I_v[i, D] = \begin{cases} \infty & \text{if } v \notin Y' \text{ and } v \in T \\ \mathcal{A}[i-1, (Y \setminus \{v\}, Y', \mathcal{P}, T')] & \text{if } v \notin Y' \text{ and } v \notin T \\ \mathcal{A}[i-1, (Y \setminus \{v\}, Y' \setminus \{v\}, \mathcal{P} \setminus \{\{v\}\}, T' \setminus \{v\})] & \text{if } v \in Y' \end{cases} \tag{9.2}$$

Intuitively, if Y is a bag corresponding to a node t in a tree decomposition of S and T' is the set of terminals in S_t, then Equation 9.2 corresponds to the computation of $c[t, Y', \mathcal{P}]$ in the dynamic programming algorithm of Steiner Tree. See [Cygan 2015, Theorem 7.8] for more detailed explanation.

For all $u, v \in Y$ and $uv \in E(G)$,

$$I_{uv}[i, D] = \min \left\{ \min_{\mathcal{P}'} \left\{ \mathcal{A}[i-1, (Y, Y', \mathcal{P}', T')] + \mathsf{recdist}(uv) \right\}, \mathcal{A}[i-1, D] \right\}, \tag{9.3}$$

where $\mathcal{P}'$ varies over partitions of Y' such that u and v are in different parts of $\mathcal{P}'$ and by merging these two parts we get the partition $\mathcal{P}$. Note that if $\{u, v\} \not\subseteq Y'$ or u and v are in same part of $\mathcal{P}$, then Equation 9.3 gives $I_{uv}[i, D] = \mathcal{A}[i-1, D]$. Equation 9.3 corresponds to the computation of values in the introduce edge node where the edge uv is introduced.

For all $w \in V(G)$,

$$F_w[i, D] = \min\{\min_{\mathcal{P}'}\{\mathcal{A}[i-1, (Y \cup \{w\}, Y' \cup \{w\}, \mathcal{P}', T')]\}, \mathcal{A}[i-1, (Y \cup \{w\}, Y', \mathcal{P}, T')]\}, \tag{9.4}$$

where $\mathcal{P}'$ in the inner minimum varies over all the partitions obtained by adding w to one of the existing parts. Equation 9.4 corresponds to computation in a forget node where w is forgotten.

$$J[i, D] = \min_{\substack{\mathcal{P}=\mathcal{P}_1 \sqcup \mathcal{P}_2 \\ T'=T'_1 \cup T'_2 \\ i' \leq i-1}} \left\{\mathcal{A}[i-1, (Y, Y', \mathcal{P}_1, T'_1)] + \mathcal{A}[i', (Y, Y', \mathcal{P}_2, T'_2)]\right\} \tag{9.5}$$

Equation 9.5 corresponds to a computation in a join node.

We define $\mathcal{A}[i, D]$ for $i \in [N] \setminus \{1\}$ and type $D = (Y, Y', \mathcal{P}, T')$ as,

$$\mathcal{A}[i, D] = \min \begin{cases} \min\limits_{v \in Y} I_v[i, D] \\ \min\limits_{\substack{uv \in E(G) \\ u,v \in Y}} I_{uv}[i, D] \\ \min\limits_{w \in V(G)} F_w[i, D] \\ J[i, D] \end{cases} \tag{9.6}$$

For each $i \in [N]$ and each type D, we associate with $A[i, D]$ a subgraph of S. We say that a subgraph F is of type $(Y, Y', \mathcal{P}, T')$, where $\mathcal{P} = \{P_1, \ldots P_q\}$ if the following holds.

(a) The number of connected components in F is equal to $|\mathcal{P}| = q$. We can order the connected components $C_1, \ldots, C_q$ of F such that $V(C_i) \cap Y = P_i$.
(b) $V(F) \cap T = T'$.

In the following lemma, we show the connection between $A[i, D]$ and a graph of type D.

Lemma 9.7. *Let $i \in [N]$ and D be a type. Furthermore, let $A[i, D]$ be computed by the Equation 9.6, and have a finite value ℓ. Then there is a subgraph F, of type D, such that* $\mathsf{recdist}(F) \leq \ell$.

Proof. We prove the statement using induction on i. Since the graph $(\{t^*\}, \emptyset)$ is of type $(\{t^*\}, \{t^*\}, \{\{t^*\}\}, \{t^*\})$, the base case holds trivially. Let $1 < i \leq N$ and $D = (Y, Y', \mathcal{P}, T')$ be a type and $\mathcal{A}[i, D] = \ell$. We need to show that there is a subgraph F of G such that F has type D and $\mathsf{recdist}((F) \leq \ell$. We know that $\mathcal{A}[i, D]$ is computed using Equation 9.6, which is a minimum over a set of values. Suppose $\mathcal{A}[i, D] = I_v[i, D] = \ell$ for some $v \in Y$. If $v \notin Y'$ and $v \notin T$, then by Equation 9.2, $\ell = I_v[i, D] = \mathcal{A}[i-1, (Y \setminus \{v\}, Y', \mathcal{P}', T')]$. By induction hypothesis there is a subgraph F of type $D' = (Y \setminus \{v\}, Y', \mathcal{P}', T')$ and $\mathsf{recdist}(F) \leq \ell$. The definition of satisfying type D or D' (conditions (a) and (b)) implies that F is of type D as well. If $v \in Y'$, then $\ell = I_v[i, D] = \mathcal{A}[i-1, D'' = (Y \setminus \{v\}, Y' \setminus \{v\}, \mathcal{P} \setminus \{v\}, T' \setminus \{v\})]$. By induction hypothesis, there is a subgraph F such that $\mathsf{recdist}(F) \leq \ell$ and F is of type D''. This implies that the graph $F' = F \cup (\{v\}, \emptyset)$ is of type D and $\mathsf{recdist}(F') = \mathsf{recdist}(F) \leq \ell$. In all other cases the proof follows by similar arguments. □

The next lemma helps us to relate an optimal rectilinear Steiner tree to the values computed for the table $\mathcal{A}[,]$. First we recall the definition of $c[,,]$. For a subset $X \subseteq X_t$, and a partition $\mathcal{P}$ of X with q parts, let $c[t, X, \mathcal{P}]$ be the minimum weight of the subgraph F of S_t such that the following hold: (i) F has q connected components $C_1, \ldots, C_q$ such that $\emptyset \neq P_i = X_t \cap V(C_i)$, (ii) $X_t \cap V(F) = X$, and (iii) $T \cap V_t \subseteq V(F)$. If there is no such subgraph F, then the value of $c[t, X, \mathcal{P}]$ is ∞.

Lemma 9.8. *Let $\mathcal{T} = (\mathbb{T}, \{X_t\}_{t \in V(\mathbb{T})})$ be a nice tree decomposition of S. For a node t, let X_t be the corresponding bag, $X \subseteq X_t$, $\mathcal{P}$ be a partition of X, V_t be the union of bags in the subtree rooted at t, and $T' = T \cap V_t$. Then $A[i, (X_t, X, \mathcal{P}, T')] \leq c[t, X, \mathcal{P}]$.*

Proof. Let $\mathcal{T} = (\mathbb{T}, \{X_t\}_{t \in V(\mathbb{T})})$ be a nice tree decomposition of S and $|V(\mathbb{T})| \leq 3(|V(S)| + |E(S)|) \leq N$. Recall that t^* is a fixed terminal. We add t^* to all the bags in $\mathcal{T}$. This new decomposition still satisfies all the property of a tree decomposition. The width of this new tree decomposition increases by at most 1. For the ease of presentation, we use $\mathcal{T} = (\mathbb{T}, \{X_t\}_{t \in V(\mathbb{T})})$ to denote the new tree decomposition of S. Note that all the leaf bags and the root bag contain only one element, t^*. Let r be the root of $\mathbb{T}$. For any node $t \in V(\mathcal{T})$ we define the *level* of t as the height of the subtree rooted at t. Recall, that the *height* of a node t is the number of vertices in the longest downward path to a leaf from that node. Note that leaves in $\mathcal{T}$ other than root r, have level 1 and the level of r is the height of $\mathbb{T}$. The level of any node in $\mathbb{T}$ is at most N. For any node $t \in V(\mathcal{T})$, we use ℓ_t to denote the level of t. For any t, we denote the graph S_t as $(V_t, \{e \mid e \text{ is introduced in a the subtree rooted at } t\})$, where V_t is the union of bags present in the subtree rooted at t.

We prove the following statement: For any $t \in V(\mathbb{T})$, $X \subseteq X_t$ and a partition $\mathcal{P}$ of X, $\mathcal{A}[\ell_t, (X_t, X, \mathcal{P}, T \cap V_t)] \leq c[t, X, \mathcal{P}]$. We prove the statement using induction on the level of the node t. The base case is when $\ell_t = 1$. In this case, $X_t = \{t^*\}$. If $X = \{t^*\}$ and $\mathcal{P} = \{\{t^*\}\}$, by definition $\mathcal{A}[1, (X_t, X, \{X\}, T \cap V_t)] = 0 = c[t, X, \{X\}]$. Otherwise, $\mathcal{A}[1, (X_t, X, \mathcal{P}, T \cap V_t)] = \infty = c[t, X, \{X\}]$. Let t be a node in $V(\mathcal{T})$, $X \subseteq X_t$ and $\mathcal{P}$ be a partition of X such that $1 < \ell_t \leq N$. Let $T' = T \cap V_t$. If $(X_t \setminus X) \cap T \neq \emptyset$, then by definition $c[t, X, \mathcal{P}] = \infty$ and so the statement holds. Suppose $(X_t \setminus X) \cap T = \emptyset$. Since $\widehat{S}$ is a Steiner tree for T, $T \subseteq V(\widehat{S})$. Since X is a bag in the tree decomposition $\mathcal{T}$, all the terminals in a connected component C of $\widehat{S} - X_t$ are either fully contained in V_t or none of the terminals in the component C are present in V_t. Thus, there are connected components $C_1, \ldots, C_j$ of $\widehat{S} - X_t$ such that $T' = T \cap V_t = T \cap (X_t \cup \bigcup_{i=1}^{j} V(C_i))$. This implies that $(X_t, X, \mathcal{P}, T')$ is a type. Let $\mathcal{P} = \{P_1, \ldots, P_q\}$ and F be a witness subgraph for the value $c[t, X, \mathcal{P}]$. That is, $\mathsf{recdist}(F) = c[t, X, \mathcal{P}]$ and the following conditions hold: (i) F has q connected components $C_1, \ldots, C_q$ such that $\emptyset \neq P_i = X_t \cap V(C_i)$, (ii) $X_t \cap V(F) = X$, and (iii) $T \cap V_t \subseteq V(F)$. We analyse cases based on the nature of the node t.

Case 1: t is an introduce vertex node. Let t' be the child of t and $\{v\} = X_t \setminus X'_t$. Note that the level of t' is $\ell_t - 1$. If $v \notin V(F)$, then $c[t', X, \mathcal{P}] \leq \mathsf{recdist}(F)$. By Equations 9.6 and 9.2, $\mathcal{A}[\ell_t, (X_t, X, \mathcal{P}, T')] \leq \mathcal{A}[\ell_t - 1, (X'_t, X, \mathcal{P}, T')]$. By induction hypothesis, $\mathcal{A}[\ell_t - 1, (X'_t, X, \mathcal{P}, T')] \leq c[t', X, \mathcal{P}] \leq \mathsf{recdist}(F) = c[t, X, \mathcal{P}]$. If $v \in V(F)$, then v appears as an isolated vertex in F, because v is an isolated vertex in S_t. This implies that $c[t', X \setminus \{v\}, \mathcal{P} \setminus \{\{v\}\}] \leq \mathsf{recdist}(F)$. By Equations 9.6 and 9.2, $\mathcal{A}[\ell_t, (X_t, X, \mathcal{P}, T')] \leq \mathcal{A}[\ell_t - 1, (X'_t, X \setminus \{v\}, \mathcal{P} \setminus \{\{v\}\}, T')]$. By induction hypothesis, $\mathcal{A}[\ell_t - 1, (X'_t, X \setminus \{v\}, \mathcal{P} \setminus \{\{v\}\}, T')] \leq c[t', X \setminus \{v\}, \mathcal{P} \setminus \{\{v\}\}] \leq \mathsf{recdist}(F \setminus \{v\}) = \mathsf{recdist}(F) = c[t, X, \mathcal{P}]$.

Case 2: t is an introduce edge node. Let t be labelled with the edge uv and t' be the child of t. That is, $\{u, v\} \subseteq X_{t'} = X_t$. Note that the level of t' is $\ell_t - 1$. If $uv \notin E(F)$, then $c[t', X, \mathcal{P}] \leq \mathsf{recdist}(F)$. By Equations 9.6 and 9.3 we know that $\mathcal{A}[\ell_t, (X_t, X, \mathcal{P}, T')] \leq \mathcal{A}[\ell_t - 1, (X'_t, X, \mathcal{P}, T')]$. By induction hypothesis, $\mathcal{A}[\ell_t - 1, (X'_t, X, \mathcal{P}, T')] \leq c[t', X, \mathcal{P}] \leq \mathsf{recdist}(F) = c[t, X, \mathcal{P}]$.

Suppose $uv \in E(F)$. Let $C'_1, \ldots, C'_{q'}$ are the connected components of $F - uv$. Consider the partition $\mathcal{P}'$ to be $\{V(C'_1) \cap X, \ldots, V(C'_{q'}) \cap X\}$. This implies that $c[t', X, \mathcal{P}'] \leq \mathsf{recdist}(F \setminus \{uv\}) = \mathsf{recdist}(F) - \mathsf{recdist}(uv)$. By induction hypothesis, $\mathcal{A}[\ell_t - 1, (X_{t'}, X, \mathcal{P}', T')] \leq c[t', X, \mathcal{P}'] \leq \mathsf{recdist}(F) - \mathsf{recdist}(uv)$. The property of F implies that in the partition $\mathcal{P}'$, if we merge the parts containing u and v, then we get the partition $\mathcal{P}$. Thus, by Equations 9.6 and 9.3, $\mathcal{A}[\ell_t, (X_t, X, \mathcal{P}, T')] \leq \mathcal{A}[\ell_t - 1, (X_{t'}, X, \mathcal{P}', T')] + \mathsf{recdist}(uv) \leq \mathsf{recdist}(F)$.

Case 3: t is a forget node. Let t' be the child of t and $\{w\} = X_{t'} \setminus X_t$. Note that the level of t' is $\ell_t - 1$. If $w \notin V(F)$, then $c[t', X, \mathcal{P}] \leq \mathsf{recdist}(F)$. By induction hypothesis, $\mathcal{A}[\ell_t - 1, (X_{t'}, X, \mathcal{P}, T)] \leq c[t', X, \mathcal{P}] \leq \mathsf{recdist}(F)$. By Equations 9.6 and 9.4, $\mathcal{A}[\ell_t, (X_t, X, \mathcal{P}, T')] \leq \mathcal{A}[\ell_t - 1, (X_{t'}, X, \mathcal{P}, T')] \leq \mathsf{recdist}(F)$.

Suppose $w \in V(F)$. Let C_i be the component of F containing w. Note that $P_i = V(C_i) \cap X$. Let $\mathcal{P}'$ be a partition obtained from $\mathcal{P}$, by adding w to the part P_i. Then $\mathcal{P}'$ is a partition of $X \cup \{w\}$. This implies that $c[t', X \cup \{w\}, \mathcal{P}'] \leq \mathsf{recdist}(F)$. By induction hypothesis, $\mathcal{A}[\ell_{t'}, (X_{t'}, X \cup \{w\}, \mathcal{P}', T')] \leq c[t', X \cup \{w\}, \mathcal{P}'] \leq \mathsf{recdist}(F)$. By Equations 9.6 and 9.4, $\mathcal{A}[\ell_t, (X_t, X, \mathcal{P}, T')] \leq \mathcal{A}[\ell_t - 1, (X_{t'}, X \cup \{w\}, \mathcal{P}', T')] \leq \mathsf{recdist}(F)$.

Case 4: t is a join node. Let t_1 and t_2 be the children of t. Here $X_t = X_{t_1} = X_{t_2}$, and the level of $X_{t_i}, i \in \{1, 2\}$, is $\ell_t - 1$. Let F_1 be the graph with vertex set as $V(F) \cap V_{t_1}$ and edge set as $E(F) \cap E(S_{t_1})$. Let F_2 be the graph with vertex set as $V(F) \cap V_{t_2}$ and edge set as $E(F) \setminus E(F_1)$. Note that $F = F_1 \cup F_2$. Let $T'_1 = V(F_1) \cap T$ and $T'_2 = V(F_2) \cap T$. Since all the connected components in $V(F)$ contain at least one vertex from X and X_t is a bag in the tree decomposition, all the connected components in F_1 as well as in F_2 contain at least one vertex from X. Let $C'_1, \ldots, C_{q'}$ be the connected components of F_1 and $C''_1, \ldots, C''_{q''}$ be the connected components in F_2. Let $\mathcal{P}_1 = \{X \cap V(C'_i), \ldots, X \cap V(C'_{q'})\}$ and $\mathcal{P}_2 = \{X \cap V(C''_i), \ldots, X \cap V(C''_{q''})\}$. Thus, $c[t_1, X, \mathcal{P}_1] \leq \mathsf{recdist}(F_1)$ and $c[t_2, X, \mathcal{P}_2] \leq \mathsf{recdist}(F_2)$. By induction hypothesis, $\mathcal{A}[\ell_{t_i}, (X_{t_i}, X, \mathcal{P}_i, T'_i)] \leq c[t_i, X, \mathcal{P}_i]$ for $i \in \{1, 2\}$. The definitions of F, F_1 and F_2 imply that $\mathcal{P} = \mathcal{P}_1 \sqcup \mathcal{P}_2$. By Equations 9.5 and 9.6, it follows that $\mathcal{A}[\ell_t, (X_t, X, \mathcal{P}, T')] \leq \mathcal{A}[\ell_t - 1, (X_t, X, \mathcal{P}_1, T'_1)] + \mathcal{A}[\ell_t - 1, (X_t, X, \mathcal{P}_2, T'_2)] \leq \mathsf{recdist}(F_1) + \mathsf{recdist}(F_2) = \mathsf{recdist}(F)$. □

Finally, we describe the subexponential algorithm for Rectilinear Steiner Tree.

Theorem 9.1. Rectilinear Steiner Tree *can be solved in time* $2^{\mathcal{O}(\sqrt{n} \log n)} n^{\mathcal{O}(1)}$.

Proof. We take as input a set T of n terminal points, the Hanan grid G of T and the weight function $\mathsf{recdist}$. Then, using Lemma 9.2, we compute a shortest path RST $\widehat{S}$. By Lemma 9.5, we know that there is an optimal Steiner tree S_{opt} with $\mathbf{tw}(\widehat{S} \cup S_{\mathsf{opt}}) \leq 41\sqrt{n}$. Based on the shortest path RST $\widehat{S}$, we apply Lemma 9.6, to enumerate all possible types D of G. We fix an integer $N = 3(|V(G)| + |E(G)|)$ and a terminal t^* in T. For each $i \in [N]$ and each type D, the algorithm computes values $\mathcal{A}[i, D]$, according to Equations 9.1 and 9.6. The values in $\mathcal{A}[,]$ are filled in the increasing order of i. Finally, the algorithm outputs $\min_{i \in [N]} \mathcal{A}[i, (\{t^*\}, \{t^*\}, \{\{t^*\}\}, T)]$.

For the hypothetical set S, fix an optimal nice tree decomposition $\mathcal{T}$, rooted at node r. Add a fixed terminal t^* to each bag of the nice tree decomposition (as described in the proof of Lemma 9.8). The treewidth of this tree decomposition is bounded by $41\sqrt{n}+1$. Let t be a node in the tree decomposition, of level ℓ_t. Let X_t be the bag of t and V_t be the union of bags in the subtree rooted at t. Let $T' = T \cap V_t$. Suppose $X \subseteq X_t$ and $\mathcal{P}$ is a partition of X. By definition, $c[t, X, \mathcal{P}]$ is the size of a subgraph of type $(X_t, X, \mathcal{P}, T')$. Then, Lemmata 9.7 and 9.8 imply that $\mathcal{A}[\ell_t, (X_t, X, \mathcal{P}, T')] = c[t, X, \mathcal{P}]$. In particular, for the root r of the tree decomposition, $\mathcal{A}[\ell_r, (\{t^*\}, \{t^*\}, \{\{t^*\}\}, T)] = c[r, \{t^*\}, \{\{t^*\}\}]$. Notice, that $c[r, \{t^*\}, \{\{t^*\}\}]$ is the size of a minimum Steiner tree of G.

On the other hand, by Lemma 9.7, for all $i \in [N]$, if $A[i, (\{t^*\}, \{t^*\}, \{\{t^*\}\}, T)] = \ell$ then there is a subgraph F that connects all the terminals of T, and that satisfies $\mathsf{recdist}(F) \leq \ell$. As the algorithm outputs $\min_{i \in [N]} \mathcal{A}[i, (\{t^*\}, \{t^*\}, \{\{t^*\}\}, T)]$, it must output the weight of a minimum rectilinear Steiner tree of G. This proves the correctness of the algorithm.

The size of the table $\mathcal{A}[,]$ is $N \cdot 2^{\mathcal{O}(\sqrt{n} \log n)}$ and each entry can be filled in time $2^{\mathcal{O}(\sqrt{n} \log n)} n^{\mathcal{O}(1)}$. Thus, the running time of the algorithm is $2^{\mathcal{O}(\sqrt{n} \log n)} n^{\mathcal{O}(1)}$. Using standard back-tracking tricks we can also output an optimal RST. This concludes the proof. □

9.4 Subexponential Algorithm for Rectilinear Steiner Arborescence

In this section, we are again given T, and the root vertex $r \in T$ as an input of Rectilinear Steiner Arborescence. Furthermore, let $|T| = n$ and G be the Hanan grid of T. We assume that the root vertex r is placed at $(0, 0)$ in $\mathbb{R}^2$. Let $\mathsf{recdist}_G$ be the weight function defined on the edges of G. In short, we use $\mathsf{recdist}$ for $\mathsf{recdist}_G$. Our aim is to design a subexponential algorithm for Rectilinear Steiner Arborescence. The algorithm is very similar to that for Rectilinear Steiner Tree. We define a rectilinear Steiner arborescence, called *shortest path RSA*, for a graph G. Then, for a shortest path RSA $\widehat{S}$, we show that there exists an optimal rectilinear Steiner arborescence S_{opt} such that the treewidth of $\widehat{S} \cup S_{\mathsf{opt}}$ is bounded. Aided by this information, we design a dynamic programming algorithm for Rectilinear Steiner Arborescence.

9.4.1 Shortest path RSA and its properties

For a given G, we define a *shortest path RSA*, which is a rectilinear Steiner Arborescence as follows.

We give an arbitrary ordering $\{r, t_1, \ldots t_k\}$ on the terminals, such that the root is the first vertex in the ordering. We define a shortest path RSA, $\widehat{S}$, through a constructive greedy process. Initially we set S_1 to a $r \cdots t_1$ monotone path. This is a rectilinear Steiner arborescence of $\{r, t_1\}$. In the i^{th} step, we compute a rectilinear Steiner arborescence S_{i+1} of $\{t_1, \ldots t_{i+1}\}$ from S_i as follows. If $t_{i+1} \in V(S_i)$, then we set $S_{i+1} = S_i$. Otherwise, for each vertex $u \in S_i$ let ℓ_u^1 be the length of a shortest $u \cdots t_{i+1}$ path. Let ℓ_u^2 be the length of the shortest $r \cdots u$ path that exists in S_i. Also, ℓ denotes the length of a shortest $r \cdots t_{i+1}$ path. Let $N \subseteq V(S_i)$ be the set of vertices such that, for each $u \in N$, $\ell_u^1 + \ell_u^2 = \ell$. Let $u^* \in N$ be a vertex for which $\ell_{u^*}^1$ is minimized. If there is only one monotone $t_{i+1} \cdots u^*$ path, then let Q be that path. Otherwise there are two monotone $t_{i+1} \cdots u^*$ paths such

that one path has a horizontal edge incident with u^* and other has a vertical edge incident with u^*. If there is a horizontal edge in S_i which is incident with u^*, then we choose Q to be the monotone $t_{i+1} \cdots u^*$ path such that the edge in Q incident with u^* is a horizontal edge. Otherwise, we choose Q to be the monotone $t_{i+1} \cdots u^*$ path such that the edge in Q incident with u^* is a vertical edge. Then, we construct S_{i+1} by adding the monotone path Q to S_i. After $n-1$ iterations, we construct a tree $\widehat{S} = S_n$ of G, which is a Steiner arborescence of T. This is our shortest path RSA.

It is possible to construct a shortest path RSA in polynomial time.

Lemma 9.9. *Given a set T of terminal points, and the Hanan grid G of T, a shortest path RSA $\widehat{S}$, of T, can be constructed in polynomial time.*

Similar to Lemma 9.3, we can give a bound on the bend vertices of a shortest path RSA.

Lemma 9.10. *The number of bend vertices in $\widehat{S}$ is at most n.*

9.4.2 Supergraph of an optimal RSA with bounded treewidth

Let T be an input set of points for Rectilinear Steiner Arborescence, $r \in T$ is a root terminal, and G is the Hanan grid of T. In this part, given a shortest path RSA $\widehat{S}$, we show the existence of an optimum rectilinear Steiner arborescence S_{opt} of T with the property that the treewidth of $\widehat{S} \cup S_{\mathsf{opt}}$ is sublinear in the number of input points. Similar to Lemma 9.4, we can show that there is an optimal rectilinear Steiner arborescence with a bounded number of bend vertices.

Lemma 9.11. *Let T be a set of n terminals in $\mathbb{R}^2$, with a root terminal $r \in T$ and G be the Hanan grid of T. Then there is an optimum rectilinear Steiner arborescence of T in G such that the number of bend vertices of the Steiner arborescence in G is at most $3n$.*

After finding a shortest path RSA, as described by Lemma 9.9, we prove the following lemma.

Lemma 9.12. *Given a set T of n points, with $r \in T$ as the root terminal, and a shortest path RSA $\widehat{S}$ of T, there is an optimal rectilinear Steiner arborescence S_{opt} of T with the property that the treewidth of $\widehat{S} \cup S_{\mathsf{opt}}$ is bounded by $41\sqrt{n}$.*

Proof. We choose an optimum rectilinear Steiner arborescence of T and prove it satisfies the required property. Among the optimum Steiner arborescences with minimum number of bend vertices we select an arborescence S_{opt} which has maximum edge intersection with $\widehat{S}$.

We show that $\widehat{S} \cup S_{\mathsf{opt}}$ has $\mathcal{O}(\sqrt{n})$ treewidth. For the sake of contradiction, suppose $\widehat{S} \cup S_{\mathsf{opt}}$ has treewidth greater than $41\sqrt{n}$. Again, we can assume that $n \geq 4$, as otherwise we can greedily find out the best rectilinear Steiner arborescence from the constant sized Hanan grid. Then, by Proposition 9.1, there is a $9\sqrt{n} \times 9\sqrt{n}$ grid H appearing as a minor in $\widehat{S} \cup S_{\mathsf{opt}}$. Let $\mathcal{P}(H) = \{C_v | v \in V(H)\}$ be a minor model of H. For a vertex $v \in V(H)$, if any vertex of C_v is a terminal vertex of G, a bend vertex of $\widehat{S}$ or a bend vertex of S_{opt}, then we mark the vertex v in H. By Lemmata 9.10 and 9.11, the number of vertices of H that get marked is at most $5n$. By the arguments given in Lemma 9.5, we can find from

H, a 2×2 grid H' in H, where none of the vertices are marked. With arguments similar to Claim 9.2, we can also find a subgrid G' in G (in some sense contained in H') that has the following properties:

1. No vertex in G' is a terminal vertex of G, or a bend vertex for $\widehat{S}$ or S_{opt}.
2. There are four vertices $u_1, \ldots, u_4$ which are of degree exactly four in $\widehat{S} \cup S_{\mathsf{opt}}$. All other vertices are of degree exactly two in $\widehat{S} \cup S_{\mathsf{opt}}$.
3. There are horizontal paths $P_{12} = u_1 \cdots u_2$, $P_{34} = u_3 \cdots u_4$, and vertical paths $P_{13} = u_1 \cdots u_3$, $P_{24} = u_2 \cdots u_4$. Each of the internal vertices of these paths are of degree 2 in the grid G.
4. Either all the horizontal paths belong to S_{opt} and not $\widehat{S}$, while all the vertical paths belong to exactly $\widehat{S}$ and not S_{opt}, or vice-versa. These are the only two possibilities.

The following observation tells us about the position of the vertices in G', with respect to the origin, where the root terminal is positioned.

Observation 9.7. *Let T be a set of terminals with the root terminal r placed at the origin. Let G be the Hanan grid of T and S_{mml} be an edge minimal Steiner arborescence for T. Let $u, v \in V(S_{\mathsf{mml}})$ be a pair of vertices with the following properties:*

- *Either $u_x = v_x \neq r_x$ and $u_y < r_y < v_y$ or $u_y = v_y \neq r_y$ and $u_x < r_x < v_x$,*
- *The monotone $u \cdots v$ path Q is a subgraph of S_{mml}.*

Then it cannot be the case that all internal vertices of Q have degree two in S_{mml}.

Proof. Without loss of generality, assume that $u_x = y_x \neq r_x$ and $u_y < r_y < v_y$ are true. For the sake of contradiction, let the monotone $u \cdots v$ path Q be a subgraph of S_{mml} such that every internal vertex of Q is degree two in S_{mml}. Let S' be the graph obtained by deleting the internal vertices and edges of Q. Since, for each $t \in T$, there is a unique $r \cdots t$ path in S_{mml}, if t is still connected to r, in S', then the $r \cdots t$ path is still a shortest $r \cdots t$ path. We show that all terminals are still connected to r in S'. This implies that S' is a Steiner arborescence, thereby contradicting the edge minimality of S_{mml}.

Suppose there is a terminal t, such that the $r \cdots t$ path Q' of S_{mml} had an edge in common with $E(Q)$. Since every internal vertex of Q is of degree two in S_{mml}, it must be the case that the entire path Q is a subpath of Q'. By definition of Q, the entire path cannot be contained in the grid defined by r and t. However, from Observation 9.5, it cannot be the case that Q is a shortest $r \cdots t$ path. Thus, for no terminal t, does the $r \cdots t$ path in S_{mml} intersect with Q. Thus, each terminal t remains connected to r in S'. Hence, we conclude that such a path Q cannot exist in an edge minimal Steiner arborescence S_{mml}. □

By definition, both $\widehat{S}$ and S_{opt} are minimal rectilinear Steiner arborescences. Then, by Observation 9.7, it follows that G' lies in one of the quadrants of $\mathbb{R}^2$. For the sake of the proof, we assume without loss of generality that G' lies in the first quadrant of $\mathbb{R}^2$. Also, without loss of generality, let the horizontal paths belong to S_{opt} and the vertical paths belong to $\widehat{S}$. Let the length of P_{12} be l. By the definition of the subgrid G', the length of P_{34} is also l. Let the length of P_{13} be p. This is also the length of P_{24}. Suppose $l > p$. Then, we consider the Steiner arborescence formed by deleting, in S_{opt}, the path P_{12} and

adding the path P_{24}. The resulting Steiner arborescence has weight strictly less than that of weight of S_{opt}. This contradicts the choice of S_{opt}. Hence, this is not possible.

Now, suppose $l \leq p$. Consider the two paths P_{13} and P_{24}. They are paths of $\widehat{S}$. By construction and by Observation 9.4, they cannot be subpaths of a path added in a single construction step, as otherwise they will not be part of a shortest path. Also by construction, one of them is added to $\widehat{S}$ before the other. Without loss of generality, let P_{13} be added before P_{24}. By construction, P_{24} is a subpath of a path P that was added in some step i, to connect a terminal t to the already constructed S_{i-1}. By definition of H' and G', this terminal must lie outside the region formed by the subgrid G'. Since P_{24} was part of a shortest path between S_{i-1} and t, t must lie on a row strictly higher than the rows of G'. Since, u_1 and u_2 are not bend vertices in $\widehat{S}$, they have neighbours u_1' and u_2' in $\widehat{S}$. Also, u_1' and u_2' are on the same row, and u_1' is on the same column as u_1 while u_2' is on the same column as u_2. Let $P_{u_1'}$ be the path from r to u_1' in S_{i-1}, $P_{u_2'}$ be the subpath of P between r and u_2', and P_t be the subpath of P between u_2' and t. By definition of a shortest path, the subpath $P_{u_2'}$ is a shortest path between r and u_2' and the subpath P_t is a shortest path between u_2' and t. Let $G_{u_1'} \leq_s G$ be the grid defined by r, u_1' as its diagonal points and $G_{u_2'} \leq_s G$ be the grid defined by r, u_2' as its diagonal points. Due to the position of the vertex r, $G_{u_1'} \leq_s G_{u_2'}$. Then, by Observation 9.5, $P_{u_1'} \cup P'$ is a shortest path between r and u_2', as is $P_{u_2'}$. This implies that $P_{u_1'} \cup P' \cup P_t$ is a shortest $r \cdots t$ path. Notice that, P is of weight at least (u_2', u_2, u_4). Since $l \leq p$, this implies that P is of weight strictly more that the path P'. However, by the description of construction of $\widehat{S}$, the path $P' \cup P_t$ is a better candidate than the path P in step i. This contradicts the choice of adding the path P to form S_i. Therefore, it is not possible that $l \leq p$.

Thus, we obtain a contradiction to the fact that $\widehat{S} \cup S_{\mathsf{opt}}$ has a grid minor greater than $9\sqrt{n}$. This proves that $\widehat{S} \cup S_{\mathsf{opt}}$ has treewidth less than $41\sqrt{n}$. □

9.4.3 Dynamic Programming Algorithm for Rectilinear Steiner Arborescence

Using the results of the previous sections we design a subexponential algorithm for Rectilinear Steiner Arborescence. By Lemma 9.12, we know that, given a shortest path RSA $\widehat{S}$, there exists an optimal Steiner arborescence S_{opt} such that the treewidth of $S = \widehat{S} \cup S_{\mathsf{opt}}$ is bounded by $41\sqrt{n}$. As in the case of the Rectilinear Steiner Tree problem, we do not know the graph S. However, we simulate a dynamic programming algorithm which contains all the information needed to compute an optimal Steiner arborescence on the tree decomposition of S.

Again, if we knew the subgraph S of G such that there is an optimal Steiner arborescence fully contained in S, we could design a dynamic programming algorithm for Rectilinear Steiner Arborescence over the tree decomposition of S. This algorithm is similar in ideas to the algorithm for Steiner Tree over a tree decomposition of a given graph. Let $(\mathbb{T}, \mathcal{X}' = \{X_t'\}_{t \in V(\mathbb{T})})$ be a nice tree decomposition of S where $\mathbb{T}$ is a rooted tree. Let the root vertex be z. From this we obtain a tree decomposition $(\mathbb{T}, \mathcal{X} = \{X_t\}_{t \in V(\mathbb{T})})$ by adding the root terminal r to each bag X_t', $t \in V(\mathbb{T})$. We continue to name a bag X_t as we named X_t', i.e., if X_t' was a leaf bag then so is X_t and so on. Notice that the treewidth of $(\mathbb{T}, \mathcal{X})$ increases by at most 1, but the root and leaf bags of $\mathbb{T}$ are identical to the singleton set $\{r\}$. For a node t, let V_t be the union

of all bags present in the subtree of $\mathbb{T}$ rooted at t. For a node t, we define a graph $G_t = (V_t, E_t = \{e \in G \ : \ e \text{ is introduced in the subtree rooted at } t\})$. A relevant definition for this problem is that of a *locally-rooted subgraph*. A forest $F \leq_s G$, with connected components $C_1, \ldots, C_q$, is called a *locally-rooted subgraph* if, each component C_i has a special vertex, or a root vertex r_i.

The important step in the algorithm for Rectilinear Steiner Arborescence is to compute the following information about locally-rooted subgraphs: for each bag X_t, $X \subseteq X_t$, a partition $\mathcal{P} = (P_1, \ldots, P_q)$ of X, and a set $X_{\mathsf{sp}} = \{r_1, \ldots, r_q\}$ such that $r_i \in P_i$, for each $i \in [q]$, the value $c[t, X, \mathcal{P}, X_{\mathsf{sp}}]$ is the minimum weight of a locally-rooted subgraph F of G_t with the following properties:

1. F has exactly q connected components $C_1, \ldots, C_q$ such that $\emptyset \neq P_i = X_t \cap V(C_i)$ for all $i \in [q]$. That is, $\mathcal{P}$ corresponds to connected components of F.
2. $X_t \cap V(F) = X$. That is, the vertices of $X_t \setminus X$ are untouched by F.
3. $T \cap V_t \subseteq V(F)$. That is, all the terminal vertices in G_t belong to F.
4. For each $i \in [q]$, and each vertex $w \in V(C_i) \setminus \{r_i\}$, the $w \cdots r_i$ path in F is a shortest path in G and there is a $r_i \cdots r$ shortest path in G that has r_i as an internal vertex.

Again, the number of parts in $\mathcal{P}$ is q, and this variable is used throughout the section to denote the number of parts of the partition in question. Suppose we know the values $c[t, X, \mathcal{P}, X_{\mathsf{sp}}]$ for each tuple $(t, X, \mathcal{P}, X_{\mathsf{sp}})$, where $t \in V(\mathbb{T}), X \subseteq X_t$, $\mathcal{P}$ is a partition of X, and X_{sp} is a set of vertices such that there is exactly one vertex from each part of $\mathcal{P}$. Then $c[z, \{r\}, \{\{r\}\}, \{r\}]$ corresponds to the weight of a minimum Steiner arborescence. Thus, knowing the function $c[\cdot]$ is enough to know the weight of an optimal rectilinear Steiner arborescence of T in S. In our case, we do not know the graph $S = \widehat{S} \cup S_{\mathsf{opt}}$ and a tree decomposition of S, but we know that the treewidth of S is at most $41\sqrt{n}$. This implies that the number of choices for bags in a tree decomposition of S is bounded by $n^{\mathcal{O}(\sqrt{n})}$. Notice that the properties 1–3 are same as the properties maintained for designing a dynamic programming algorithm for Rectilinear Steiner Tree on a tree decomposition of S. Property 4 is necessary to solve the Rectilinear Steiner Arborescence problem. Each vertex $r_i \in X_{\mathsf{sp}}$ can be thought of as a local root vertex for the component C_i of F. The intuition behind this property is as follows. Suppose F was a subgraph of an edge minimal Steiner arborescence S^* rooted at r. Also, for each $i \in [q]$, r_i is the unique vertex in C_i that has minimum distance to r in S. Then, for each $i \in [q], w \in C_i$, the $w \cdots r$ path in S^* satisfies Property 4. In particular, if S^* was an optimal Steiner arborescence, Property 4 still holds for F. The number of choices for X_{sp}, which is a subset of X, is at most $2^{\sqrt{n}}$.

In fact, if there are two vertices u, v such that there is a $u \cdots r$ shortest path in G that has v as an internal vertex, we say that u has the *shortness property* with v. Notice that the shortness property is transitive. That is, if u has the shortness property with v and w has the shortness property with u, then w has the shortness property with v. Whether a vertex u has the shortness property with v can be tested, by checking if, in G, the length of a shortest $u \cdots v$ path plus the length of a shortest $v \cdots r$ path equals the length of a shortest $u \cdots r$ path.

We explain the algorithm more formally. We first define a *type* which is the analogue of a tuple $(t, X, \mathcal{P}, X_{\mathsf{sp}})$ in the normal dynamic programming for Rectilinear Steiner Arborescence.

Definition 9.9. *A type is a tuple $(Y, Y' \subseteq Y, \mathcal{P}, Y_{\mathsf{sp}}, T')$ such that the following holds.*

(i) Y is a subset of $V(G)$ of size at most $41\sqrt{n}+2$.
(ii) $\mathcal{P}$ is a partition of Y'.
(iii) There exists a set of components $C_1, \ldots, C_q$ in $\widehat{S} \backslash Y$ such that $T' = T \cap (Y' \cup \bigcup_{i=1}^{q} V(C_i))$.
(iv) Y_{sp} has exactly one vertex r_i from each part $P_i \in \mathcal{P}$.

In a type $(Y, Y', \mathcal{P}, Y_{\mathsf{sp}}, T')$, Y represents a potential bag Y of a node (say t) in a tree decomposition of S. The set Y', partition $\mathcal{P}$ and Y_{sp} have the same meaning as that of $(t, Y', \mathcal{P}, Y_{\mathsf{sp}})$ in the normal dynamic programming algorithm for Rectilinear Steiner Arborescence over a tree decomposition of an input graph. The set T' is the set of terminals in the graph G_t. We can show that the number of types is bounded.

Lemma 9.13. *There is a $2^{\mathcal{O}(\sqrt{n}\log n)} n^{\mathcal{O}(1)}$ time algorithm $\mathcal{B}$ enumerating all the types.*

Proof. The number of choices for Y is $n^{\sqrt{n}}$. Once a Y is fixed, the number of choices for Y' is $\mathcal{O}(2^{\sqrt{n}})$, while the number of choices for the partition $\mathcal{P}$, of Y', is $\mathcal{O}(\sqrt{n}^{\sqrt{n}})$. By definition of the Hanan grid G, each vertex in G has at most 4 neighbours. Thus, $\widehat{S} - Y$ has at most $4\sqrt{n}$ components. On fixing a Y, the choices for T' correspond to the $2^{\mathcal{O}(\sqrt{n})}$ choices of at most $4\sqrt{n}$ components of $\widehat{S} - Y$. This gives us the desired bound on the number of suitable types. □

Below, we explain the steps of our algorithm. We fix an integer $N = |3(V(G)| + |E(G)|)$. Just like the dynamic programming algorithm for Rectilinear Steiner Tree, this algorithm computes values $\mathcal{A}[i, D]$, where $i \in [N]$ and D is a type. As before, we want the table $\mathcal{A}[,]$ to contain all the information that is necessary for the correctness of the dynamic programming algorithm for Steiner Arborescence over a tree decomposition of S. Since we do not know the graph S, we assume a hypothetical nice tree decomposition $\mathcal{T} = (\mathbb{T}, \mathcal{X} = \{X_t\}_{t \in V(\mathbb{T})})$ of S. We may also assume that the nice tree decomposition is rooted at a node $z \in \mathbb{T}$. The level of a vertex $t \in \mathbb{T}$ is the height of the subtree of $\mathbb{T}$ rooted at t. The level of any node of $\mathbb{T}$ is at most N. Suppose $t \in \mathbb{T}$ is a node at level i, and corresponds to the bag X_t. Let V_t be the union of bags present in the subtree rooted at t. Let the graph G_t be defined as $(V_t, \{e \mid e \text{ is introduced in the subtree rooted at } t\})$. Let $T' = V_t \cap T$. Then, for any $X \subseteq X_t$, a partition $\mathcal{P}$ of X, and a set X_{sp} defined with exactly one vertex from each part of $\mathcal{P}$, $\mathcal{A}[i, (X_t, X, \mathcal{P}, X_{\mathsf{sp}}, T')] = c[t, X, \mathcal{P}, X_{\mathsf{sp}}]$. As mentioned before, $c[t, X, \mathcal{P}, X_{\mathsf{sp}}]$ is the minimum weight of the locally-rooted subgraph F of G_t such that the following hold: (i) F has q connected components $C_1, \ldots, C_q$ such that $\emptyset \neq P_i = X_t \cap V(C_i)$, (ii) $X_t \cap V(F) = X$, (iii) $T \cap V_t \subseteq V(F)$, and (iv) for each $i \in [q]$, and each vertex $w \in V(C_i) \setminus \{r_i\}$, the $w \cdots r_i$ path in F is a shortest path in G and w has the shortness property with r_i. For other pairs (i, D), we do not guarantee that the value of $\mathcal{A}[i, D]$ is meaningful. As we had motivated the ideas behind the algorithm for Rectilinear Steiner Tree, the main idea behind this algorithm is that we pretend that a tree decomposition for S is given to us, even though in reality we do not even know the graph S. Our dynamic programming is designed to mimic a dynamic programming algorithm for Steiner Arborescence over a tree decomposition of an input graph.

We write a recurrence relation for $\mathcal{A}[i, D]$, where $i \in [N]$ and D is a type. The motivation for the recurrence relation is similar to that for the recurrence in the subexponential

algorithm of Rectilinear Steiner Tree.

$$\mathcal{A}[1, D] = \begin{cases} 0 & \text{if } D = (\{r\}, \{r\}, \{\{r\}\}, \{r\}, \{r\}) \\ \infty & \text{otherwise} \end{cases} \tag{9.7}$$

In order to define $\mathcal{A}[i, D]$ for $i \in [N] \setminus \{1\}$ and a type $D = (Y, Y', \mathcal{P}, Y_{\mathsf{sp}}, T')$, we first define many intermediate values and take the minimum over all such values.

For all $v \in Y$,

$$I_v[i, D] = \begin{cases} \infty & \text{if } v \notin Y' \\ & \text{and } v \in T \\ \infty & \text{if } v \in Y' \\ & \text{but } \{v\} \notin \mathcal{P} \\ \mathcal{A}[i-1, (Y \setminus \{v\}, Y', \mathcal{P}, Y_{\mathsf{sp}}, T'] & \text{if } v \notin Y' \\ & \text{and } v \notin T \\ \mathcal{A}[i-1, (Y \setminus \{v\}, Y' \setminus \{v\}, \mathcal{P} \setminus \{\{v\}\}, Y_{\mathsf{sp}} \setminus \{v\}, T' \setminus \{v\})] & \text{if } v \in Y' \end{cases} \tag{9.8}$$

Intuitively, if Y is a bag corresponding to a node t in a tree decomposition of S and T' is the set of terminals in G_t, then Equation 9.8 corresponds to computation of $c[t, Y', \mathcal{P}, Y_{\mathsf{sp}}]$ in the dynamic programming algorithm of Rectilinear Steiner Arborescence.

For all $u, v \in Y$ such that u, v belong to the same part P of $\mathcal{P}$ and $uv \in E(G)$, we do the following. Let w be the vertex that belongs to $P \cap Y_{\mathsf{sp}}$. We first define a set $\mathbb{P}$ of pairs $(\mathcal{P}', Y'_{\mathsf{sp}})$ on Y'. For a pair $(\mathcal{P}', Y'_{\mathsf{sp}})$, $\mathcal{P}'$ denotes a partition of Y', while Y'_{sp} is defined by taking exactly one vertex per part of $\mathcal{P}'$. For a pair $(\mathcal{P}', Y'_{\mathsf{sp}})$ to belong to $\mathbb{P}$ they must be exactly one of the following kinds of pairs:

1. The vertices u, v belong to the same part of $\mathcal{P}'$. For each part $P' \in \mathcal{P}'$, and vertex $v \in P' \cap Y'_{\mathsf{sp}}$, every vertex of P' has the shortness property with v.
2. The vertices u, v belong to distinct parts P_u, P_v, respectively, of $\mathcal{P}'$. For each part $P' \in \mathcal{P}'$, and vertex $v \in P' \cap Y'_{\mathsf{sp}}$, every vertex of P' has the shortness property with v. Assume $u^* \in Y'_{\mathsf{sp}} \cap P_u$ and $v^* \in Y'_{\mathsf{sp}} \cap P_v$. Then, exactly one of the two possibilities holds:
   - It holds that $u^* = w$, $v^* = v$. The vertex v has the shortness property with u, and therefore the shortness property with w.
   - It holds that $v^* = w$, $u^* = v$. The vertex u has the shortness property with v, and therefore the shortness property with w.

Then, for the pair $u, v \in Y$,

$$I_{uv}[i, D] = \min \left\{ \min_{(\mathcal{P}', Y'_{\mathsf{sp}}) \in \mathbb{P}} \left\{ \mathcal{A}[i-1, (Y, Y', \mathcal{P}', Y'_{\mathsf{sp}}, T')] + \mathsf{recdist}(uv) \right\}, \mathcal{A}[i-1, D] \right\} \tag{9.9}$$

Note that if $\{u, v\} \not\subseteq Y'$ or u and v are in same part of $\mathcal{P}$, then Equation 9.9 gives $I_{uv}[i, D] = \mathcal{A}[i-1, D]$. Equation 9.9 corresponds to the computation of values in the introduce edge node where the edge uv is introduced.

For all $w \in V(G)$,

$$F_w[i, D] = \min \Bigg\{ \min_{\mathcal{P}'} \Big\{ \mathcal{A}[i-1, (Y \cup \{w\}, Y' \cup \{w\}, \mathcal{P}', Y_{\mathsf{sp}}, T')] \Big\}, \mathcal{A}[i-1, (Y \cup \{w\}, Y', \mathcal{P}, Y_{\mathsf{sp}}, T')] \Bigg\}, \quad (9.10)$$

where $\mathcal{P}'$ in the inner minimum varies over all the partitions obtained by adding w to one of the existing parts, and w was not a local root vertex. Equation 9.10 corresponds to computation in a forget node where w is forgotten.

Let $\mathbb{Q}$ be the set of tuples, $(\mathcal{P}_1, \mathcal{P}_2, Y^1_{\mathsf{sp}}, Y^2_{\mathsf{sp}})$, that satisfies the following properties:

1. $\mathcal{P}_1 \sqcup \mathcal{P}_2 = \mathcal{P}$.
2. $Y_{\mathsf{sp}} \subseteq Y^1_{\mathsf{sp}} \cup Y^2_{\mathsf{sp}}$.
3. For $i \in [2]$, Y^i_{sp} is defined with exactly one vertex from each part of $\mathcal{P}_i$. All vertices in a part of $\mathcal{P}_i$ have the shortness property with the vertex of Y^i_{sp} in that part.
4. Let P_1 and P_2 be parts of $\mathcal{P}_1$ and $\mathcal{P}_2$ respectively. Then $|P_1 \cap P_2| \leq 1$. A vertex in $P_1 \cap P_2$ belongs to at least one of Y^1_{sp} and Y^2_{sp}. We call such a vertex an intersection vertex.
5. For any part $P \in \mathcal{P}$, let it be formed by $\{P_{11}, P_{12}, \ldots, P_{1a}\} \in \mathcal{P}_1$ and $\{P_{21}, \ldots, P_{2b}\} \in \mathcal{P}_2$. Let Y'_1 be the subset of Y^1_{sp} defined by $\{P_{11}, P_{12}, \ldots, P_{1a}\}$. Let Y'_2 be the subset of Y^2_{sp} defined by $\{P_{21}, P_{22}, \ldots, P_{2b}\}$. Let $r_P = P \cap Y_{\mathsf{sp}}$. Consider the auxiliary bipartite graph $H = (A \uplus B, E(H))$ such that the vertices in A correspond to the parts $\{P_{11}, P_{12}, \ldots, P_{1a}, P_{21}, P_{22}, \ldots, P_{2b}\}$ and $B = Y'_1 \cup Y'_2$. An edge is added between a vertex $u \in A$ and $v \in B$, if part corresponding to u contains the intersection vertex corresponding to v. This auxiliary graph H must be a tree, in order for $(\mathcal{P}_1, \mathcal{P}_2, Y^1_{\mathsf{sp}}, Y^2_{\mathsf{sp}})$ to be a tuple of $\mathbb{Q}$. For a vertex $v \in B$, let R_v be the $r_P \cdots v$ path in H. Let $L_v = \{v = v_1, v_2, \ldots, r_P = v_\ell\}$ be the sequence of intersection vertices obtained from R_v. Then for two consecutive vertices $\{v_j, v_{j+1}\}$ in L_v, v_j has the shortness property with v_{j+1}.

With respect to the set $\mathbb{Q}$, we define the following equation:

$$J[i, D] = \min_{\substack{\mathcal{P}=\mathcal{P}_1 \sqcup \mathcal{P}_2 \\ (\mathcal{P}_1, \mathcal{P}_2, Y^1_{\mathsf{sp}}, Y^2_{\mathsf{sp}}) \in \mathbb{Q} \\ T'_1 \cup T'_2 = T' \\ i' \leq i-1}} \Big\{ \mathcal{A}[i-1, (Y, Y', \mathcal{P}_1, Y^1_{\mathsf{sp}}, T'_1)] + \mathcal{A}[i', (Y, Y', \mathcal{P}_2, Y^2_{\mathsf{sp}}, T'_2)] \Big\} \quad (9.11)$$

Equation 9.11 corresponds to a computation in a join node.

We define $\mathcal{A}[i, D]$ for $i \in [N] \setminus \{1\}$ and type $D = (Y, Y', \mathcal{P}, Y_{\mathsf{sp}}, T')$ as,

$$\mathcal{A}[i, D] = \min \begin{cases} \min_{v \in Y} I_v[i, D] \\ \min_{\substack{uv \in E(G) \\ u,v \in Y}} I_{uv}[i, D] \\ \min_{w \in V(G)} F_w[i, D] \\ J[i, D] \end{cases} \quad (9.12)$$

For each $i \in [N]$ and each type D, we associate with $A[i, D]$ a locally-rooted subgraph of S. We say that a locally-rooted subgraph F is of type $(Y, Y', \mathcal{P}, Y_{\mathsf{sp}}, T')$, where $\mathcal{P} = \{P_1, \dots P_q\}$ if the following holds.

(a) The number of connected components in F is equal to $|\mathcal{P}| = q$. Also, $V(C_i) \cap Y = P_i$.
(b) $V(F) \cap T = T'$.
(c) For each $i \in [q]$, $r_i \in P_i$.
(d) For each $i \in [q]$, and each $w \in C_i$, the $r_i \cdots w$ path in F is a shortest path in G and there is a shortest $r \cdots w$ path in G with r_i appearing as an internal vertex.

The next Lemma shows the relation between the function $\mathcal{A}[\cdot]$ and the set of locally-rooted subgraphs of S.

Lemma 9.14. *Let $i \in [N]$ and D be a type. Furthermore, let $A[i, D]$ be computed by the Equation 9.12, and have a finite value ℓ. Then there is a locally-rooted subgraph F, of type D, such that* $\mathsf{recdist}(F) \leq \ell$.

Proof. We show the statement using induction on i.

Case 1: Base case. Since the graph $(\{r\}, \emptyset)$ is of type $(\{r\}, \{r\}, \{\{r\}\}, \{r\}, \{r\})$, the base case holds trivially.

Now, let $1 < i \leq N$ and $D = (Y, Y', \mathcal{P}, Y_{\mathsf{sp}}, T')$ be a type and $\mathcal{A}[i, D] = \ell$. We need to show that there is a locally-rooted subgraph F of G such that F has type D and $\mathsf{recdist}(F) \leq \ell$.

Case 2. We know that $\mathcal{A}[i, D]$ is computed using Equation 9.12, which is a minimum over a set of values. Suppose $\mathcal{A}[i, D] = I_v[i, D] = \ell$ for some $v \in Y$. If $v \notin Y'$ and $v \notin T$, then by Equation 9.8, $\ell = I_v[i, D] = \mathcal{A}[i-1, (Y \setminus \{v\}, Y', \mathcal{P}', Y_{\mathsf{sp}}, T')]$. By induction hypothesis there is a locally-rooted subgraph F which is of type $D' = (Y \setminus \{v\}, Y', \mathcal{P}', Y_{\mathsf{sp}}, T')$ and $\mathsf{recdist}(F) \leq \ell$. The definition of satisfying type D or D' (conditions (a) - (d)) implies that F is of type D as well. If $v \in Y'$, then $\ell = I_v[i, D] = \mathcal{A}[i-1, D'' = (Y \setminus \{v\}, Y' \setminus \{v\}, \mathcal{P} \setminus \{v\}, Y_{\mathsf{sp}} \setminus \{v\}, T' \setminus \{v\})]$. By induction hypothesis, there is a locally-rooted subgraph F such that $\mathsf{recdist}(F) \leq \ell$ and F is of type D''. This implies that the graph $F' = F \cup (\{v\}, \emptyset)$ is of type D and $\mathsf{recdist}(F') = \mathsf{recdist}(F) \leq \ell$.

Case 3. Suppose $\mathcal{A}[i, D] = I_{uv}[i, D] = \ell$ for some $u, v \in Y$, such that $uv \in E(G)$. If u and v are in the same part of $\mathcal{P}$, then $\mathcal{A}[i, D] = \mathcal{A}[i-1, D] = \ell$. By induction hypothesis there is a locally-rooted subgraph F which is of type D and $\mathsf{recdist}(F) \leq \ell$. On the other hand, suppose u and v are in different parts of $\mathcal{P}$. If $\mathcal{A}[i, D] = \mathcal{A}[i-1, D] = \ell$, then again by induction hypothesis we have a locally-rooted subgraph of type D and weight at most ℓ. Otherwise, $\mathcal{A}[i, D] = \mathcal{A}[i-1, (Y, Y', \mathcal{P}', Y'_{\mathsf{sp}}, T')] + \mathsf{recdist}(uv)$ for a pair $(\mathcal{P}', Y'_{\mathsf{sp}}) \in \mathbb{P}$. By induction hypothesis, there is a locally-rooted subgraph F' with type $D' = (Y, Y', \mathcal{P}', Y'_{\mathsf{sp}}, T')$, such that $\mathsf{recdist}(F') \leq \ell - \mathsf{recdist}(uv)$ and there is no $u \cdots v$ path in F'. By definition of pairs in $\mathbb{P}$ and transitivity of the shortness property, the graph $F = F' \cup (\{u, v\}, \{uv\})$ is a locally-rooted subgraph that has type D, by satisfying all of the properties (a)–(d). Since $\mathsf{recdist}(F) \leq \ell$ we are done.

Case 4. When $\mathcal{A}[i, D] = F_w[i, D] = \ell$, for some $w \in V(G)$, then the arguments are similar to the Case 1.

Case 5. Suppose $\mathcal{A}[i, D] = \mathcal{A}[i-1, (Y, Y', \mathcal{P}_1, Y^1_{\mathsf{sp}}, T'_1)] + \mathcal{A}[i-1, (Y, Y', \mathcal{P}_2, Y^2_{\mathsf{sp}}, T'_2)] = \ell$, for a tuple $(\mathcal{P}_1, \mathcal{P}_2, Y^1_{\mathsf{sp}}, Y^2_{\mathsf{sp}}) \in \mathbb{Q}$ and for $T'_1 \cup T'_2 = T'$. By induction hypothesis, for $i \in [2]$, there is a locally-rooted subgraph F_i of type $D_i = (Y, Y', \mathcal{P}_i, Y^i_{\mathsf{sp}}, T'_i)$. By the last two properties of the tuple $(\mathcal{P}_1, \mathcal{P}_2, Y^1_{\mathsf{sp}}, Y^2_{\mathsf{sp}})$, if F_1 and F_2 are forests, then $F_1 \cup F_2$ is also a forest. Also, by transitivity of the shortness property, it follows that $F = F_1 \cup F_2$ is a locally-rooted subgraph of type D. Since, $\mathsf{recdist}(F) \leq \mathsf{recdist}(F_1) + \mathsf{recdist}(F_2) \leq \ell$, this proves the hypothesis. □

The next Lemma links an optimal rectilinear Steiner arborescence to the values computed for the table $\mathcal{A}[,]$. First we recall the definition of $c[,,]$. For a subset $X \subseteq X_t$, a partition $\mathcal{P}$ of X, and a set X_{sp} defined by selecting exactly one vertex from each part of $\mathcal{P}$, let $c[t, X, \mathcal{P}, X_{\mathsf{sp}}]$ be the minimum weight of the subgraph F of G_t such that the following hold: (i) F has q connected components $C_1, \ldots, C_q$ such that $\emptyset \neq P_i = X_t \cap V(C_i)$, (ii) $X_t \cap V(F) = X$, (iii) $T \cap V_t \subseteq V(F)$, and (iv) for each $i \in [q]$, and each vertex $w \in V(C_i) \setminus \{r_i\})$, the $w \cdots r_i$ path in F is a shortest path in G and w has the shortness property with r_i. If there is no such subgraph F, then the value $c[t, X, \mathcal{P}, X_{\mathsf{sp}}]$ is ∞.

Lemma 9.15. *Let $\mathcal{T} = (\mathbb{T}, \{X_t\}_{t \in V(\mathbb{T})})$ be a nice tree decomposition of S. For a node t, let X_t be the corresponding bag, $X \subseteq X_t$, $\mathcal{P}$ be a partition of X and X_{sp} be a set defined by selecting exactly one vertex from each part of $\mathcal{P}$. Let V_t be the union of bags in the subtree rooted at t, and $T' = T \cap V_t$. Then $A[i, (X_t, X, \mathcal{P}, X_{\mathsf{sp}}, T')] \leq c[t, X, \mathcal{P}, X_{\mathsf{sp}}]$.*

Proof. Consider a nice tree decomposition $\mathcal{T} = (\mathbb{T}, \{X'_t\}_{t \in V(\mathbb{T})})$, of S, rooted at a node z. To each bag in $\mathcal{T}$ we add the root terminal r, thereby obtaining a new tree decomposition $\mathcal{T}' = (\mathbb{T}, \{X'_t\}_{t \in V(\mathbb{T})})$. The treewidth of the new tree decomposition is at most 1 more than that of the old tree decomposition. For the ease of presentation, we use $\mathcal{T} = (\mathbb{T}, \{X_t\}_{t \in V(\mathbb{T})})$ to denote the new tree decomposition of S. Note that all the leaf bags and the root bag z, of $\mathcal{T}$, contain only one element r. As before, for any node $t \in V(\mathcal{T})$ the *level* of t is the height of the subtree rooted at t. Note that leaves in $\mathcal{T}$ other than root z, have level 1. The level of z is the height of $\mathbb{T}$. The level of any node in $\mathbb{T}$ is at most N. For any node $t \in V(\mathcal{T})$ we use ℓ_t to denote the level of t. For any t we denote the graph S_t as $(V_t, \{e \mid e \text{ is introduced in the subtree rooted at } t\})$ where V_t is the union of bags present in the subtree rooted at t.

We prove the following statement : For any $t \in V(\mathbb{T})$, $X \subseteq X_t$, a partition $\mathcal{P} = \{P_1, \ldots, P_q\}$ of X, and a set $X_{\mathsf{sp}} = \{r_1, \ldots, r_q\}$ such that $r_i \in P_i$, it must be the case that $\mathcal{A}[\ell_t, (X_t, X, \mathcal{P}, X_{\mathsf{sp}}, T \cap V_t)] \leq c[t, X, \mathcal{P}, X_{\mathsf{sp}}]$. We prove the statement using induction on the level of the node t. The base case is when $\ell_t = 1$. In this case $X = \{r\}$. If $X = \{r\}$ and $\mathcal{P} = \{\{r\}\}$, by definition $\mathcal{A}[1, (X_t, X, \{X\}, X, T \cap V_t)] = 0 = c[t, X, \{X\}, X]$. Otherwise $\mathcal{A}[1, (X_t, X, \mathcal{P}, X, T \cap V_t)] = \infty = c[t, X, \{X\}, X]$. Let t be a node in $V(\mathcal{T})$, $X \subseteq X_t$, $\mathcal{P}$ be a partition of X, X_{sp} be a set defined by selecting exactly one vertex from each part of $\mathcal{P}$, and $1 < \ell_t \leq N$. Let $T' = T \cap V_t$. If $(X_t \setminus X) \cap T \neq \emptyset$, then by definition $c[t, X, \mathcal{P}, X_{\mathsf{sp}}] = \infty$ and so the statement holds. Suppose $(X_t \setminus X) \cap T = \emptyset$. Since $\widehat{S}$ is a Steiner arborescence for T, $T \subseteq V(\widehat{S})$. Since X_t is a bag in the tree decomposition $\mathcal{T}$, each terminal in a connected component C of $\widehat{S} - X_t$ is either fully contained in V_t or none of the terminals in the component C are present in V_t. Thus there exists a set of connected components $C_1, \ldots, C_j$ of $\widehat{S} - X_t$ such that $T' = T \cap V_t = T \cap (X_t \cup \bigcup_{i=1}^{j} V(C_i))$. This implies that $(X_t, X, \mathcal{P}, X_{\mathsf{sp}}, T')$ is a type. Let $\mathcal{P} = \{P_1, \ldots, P_q\}$, $X_{\mathsf{sp}} = \{r_1, \ldots, r_q\}$ such that $r_i \in P_i$, and F be a witness subgraph for the value $c[t, X, \mathcal{P}, X_{\mathsf{sp}}]$. That is,

$\mathsf{recdist}(F) = c[t, X, \mathcal{P}, X_{\mathsf{sp}}]$ and the following conditions hold: (i) F has q connected components $C_1, \ldots, C_q$ such that $\emptyset \neq P_i = X_t \cap V(C_i)$, (ii) $X_t \cap V(F) = X$, (iii) $T \cap V_t \subseteq V(F)$, and (iv) for each $i \in [q]$, and each vertex $w \in V(C_i) \setminus \{r_i\})$, the $w \cdots r_i$ path in F is a shortest path in G and w has the shortness property with r_i. We look at cases based on the nature of the node t.

Case 1: t is an introduce vertex node. Let t' be the child of t and $\{v\} = X_t \setminus X'_t$. Note that the level of t' is $\ell_t - 1$. If $v \notin V(F)$, then $c[t', X, \mathcal{P}, X_{\mathsf{sp}}] \leq \mathsf{recdist}(F)$. By Equations 9.8 and 9.12, $\mathcal{A}[\ell_t, (X_t, X, \mathcal{P}, X_{\mathsf{sp}}, T')] \leq \mathcal{A}[\ell_t - 1, (X'_t, X, \mathcal{P}, X_{\mathsf{sp}}, T')]$. By induction hypothesis,

$$\mathcal{A}[\ell_t - 1, (X'_t, X, \mathcal{P}, X_{\mathsf{sp}}, T')] \leq c[t', X, \mathcal{P}, X_{\mathsf{sp}}] \leq \mathsf{recdist}(F) = c[t, X, \mathcal{P}, X_{\mathsf{sp}}].$$

If $v \in V(F)$, then v appears as an isolated vertex in F, because v is an isolated vertex in S_t. By definition of X_{sp}, v must belong to X_{sp}. This implies that $c[t', X \setminus \{v\}, \mathcal{P} \setminus \{\{v\}\}, X_{\mathsf{sp}} \setminus \{v\}] \leq \mathsf{recdist}(F)$. By Equations 9.8 and 9.12,

$$\mathcal{A}[\ell_t, (X_t, X, \mathcal{P}, X_{\mathsf{sp}}, T')] \leq \mathcal{A}[\ell_t - 1, (X'_t, X \setminus \{v\}, \mathcal{P} \setminus \{\{v\}\}, X_{\mathsf{sp}} \setminus \{v\}, T')].$$

By induction hypothesis, $\mathcal{A}[\ell_t - 1, (X'_t, X \setminus \{v\}, \mathcal{P} \setminus \{\{v\}\}, X_{\mathsf{sp}} \setminus \{v\}, T')] \leq c[t', X \setminus \{v\}, \mathcal{P} \setminus \{\{v\}\}, X_{\mathsf{sp}} \setminus \{v\}] \leq \mathsf{recdist}(F \setminus \{v\}) = \mathsf{recdist}(F) = c[t, X, \mathcal{P}, X_{\mathsf{sp}}]$.

Case 2: t is an introduce edge node. Let t be labelled with the edge uv and t' be the child of t. That is, $\{u, v\} \subseteq X_{t'} = X_t$. Note that the level of t' is $\ell_t - 1$. If $uv \notin E(F)$, then $c[t', X, \mathcal{P}, X_{\mathsf{sp}}] \leq \mathsf{recdist}(F)$. By Equations 9.9 and 9.12 we know that

$$\mathcal{A}[\ell_t, (X_t, X, \mathcal{P}, X_{\mathsf{sp}}, T')] \leq \mathcal{A}[\ell_t - 1, (X'_t, X, \mathcal{P}, X_{\mathsf{sp}}, T')].$$

By induction hypothesis,

$$\mathcal{A}[\ell_t - 1, (X'_t, X, \mathcal{P}, X_{\mathsf{sp}}, T')] \leq c[t', X, \mathcal{P}, X_{\mathsf{sp}}] \leq \mathsf{recdist}(F) = c[t, X, \mathcal{P}, X_{\mathsf{sp}}].$$

Suppose $uv \in E(F)$. This means that there is a single component C, in F, that contains u, v. Let $r_C = C \cap X_{\mathsf{sp}}$. Then, for each vertex $w \in C$, w has the shortness property with r_C. Let $C'_1, \ldots, C'_{q'}$ be the connected components of $F - uv$. Since each component of F is a tree, the component C breaks into two components, C' and C'', of $F \setminus \{uv\}$. Without loss of generality, assume that $r_C, u \in C'$ and $v \in C''$. Notice that any vertex of C'' has the shortness property with v, and any vertex of C' continues to have the shortness property with r_C. Consider the partition $\mathcal{P}'$ to be $\{V(C'_1) \cap X, \ldots, V(C'_{q'}) \cap X\}$ and $X'_{\mathsf{sp}} = X_{\mathsf{sp}} \cup \{v\}$. This implies that $c[t', X, \mathcal{P}', X'_{\mathsf{sp}}] \leq \mathsf{recdist}(F \setminus \{uv\}) = \mathsf{recdist}(F) - \mathsf{recdist}(uv)$. By induction hypothesis,

$$\mathcal{A}[\ell_t - 1, (X_{t'}, X, \mathcal{P}', X'_{\mathsf{sp}}, T')] \leq c[t', X, \mathcal{P}', X_{\mathsf{sp}}] \leq \mathsf{recdist}(F) - \mathsf{recdist}(uv).$$

The property of F implies that in the partition $\mathcal{P}'$, if we merge the parts containing u and v, then we get the partition $\mathcal{P}$. This, along with the definition of X'_{sp}, implies that the tuple $(\mathcal{P}', X'_{\mathsf{sp}}) \in \mathbb{P}$. Thus, by Equations 9.9 and 9.12,

$$\mathcal{A}[\ell_t, (X_t, X, \mathcal{P}, X_{\mathsf{sp}}, T')] \leq \mathcal{A}[\ell_t - 1, (X_{t'}, X, \mathcal{P}', X'_{\mathsf{sp}}, T')] + \mathsf{recdist}(uv) \leq \mathsf{recdist}(F).$$

Case 3: t is a forget node. Let t' be the child of t and $\{w\} = X_{t'} \setminus X_t$. Note that the level of t' is $\ell_t - 1$. If $w \notin V(F)$, then $c[t', X, \mathcal{P}, X_{\mathsf{sp}}] \leq \mathsf{recdist}(F)$. By induction

hypothesis, $\mathcal{A}[\ell_t,(X_{t'},X,\mathcal{P},X_{\mathsf{sp}},T)] \leq c[t',X,\mathcal{P},X_{\mathsf{sp}}] \leq \mathsf{recdist}(F)$. By Equations 9.10 and 9.12, $\mathcal{A}[\ell_t,(X_t,X,\mathcal{P},X_{\mathsf{sp}},T')] \leq \mathcal{A}[\ell_t-1,(X_{t'},X,\mathcal{P},X_{\mathsf{sp}},T')] \leq \mathsf{recdist}(F)$.

Suppose $w \in V(F)$. Note that w does not belong to X. Consider the set $X \cup \{w\} \subseteq X_{t'}$. Let C_i be the component of F containing w. Note that $P_i = V(C_i) \cap X$. Since, $w \notin X$, $w \neq C_i \cap X_{\mathsf{sp}}$. Let $\mathcal{P}'$ is a partition obtained from $\mathcal{P}$, by adding w to the part P_i. Then $\mathcal{P}'$ is a partition of $X \cup \{w\}$. This implies that $c[t',X\cup\{w\},\mathcal{P}',X_{\mathsf{sp}}] \leq \mathsf{recdist}(F)$. By induction hypothesis,

$$\mathcal{A}[\ell_{t'},(X_{t'},X\cup\{w\},\mathcal{P}',X_{\mathsf{sp}},T')] \leq c[t',X\cup\{w\},\mathcal{P}',X_{\mathsf{sp}}] \leq \mathsf{recdist}(F).$$

By Equations 9.10 and 9.12,

$$\mathcal{A}[\ell_t,(X_t,X,\mathcal{P},X_{\mathsf{sp}},T')] \leq \mathcal{A}[\ell_t-1,(X_{t'},X\cup\{w\},\mathcal{P}',X_{\mathsf{sp}},T')] \leq \mathsf{recdist}(F).$$

Case 4: t is a join node. Let t_1 and t_2 be the children of t. Here $X_t = X_{t_1} = X_{t_2}$ and the level of $X_{t_i}, i \in \{1,2\}$, is $\ell_t - 1$. Let F_1 be the graph with vertex set $V(F) \cap V_{t_1}$ and edge set $E(F) \cap E(S_{t_1})$. Let F_2 be the graph with vertex set $V(F) \cap V_{t_2}$ and edge set $E(F) \setminus E(F_1)$. As F was a forest, the graphs F_1 and F_2 are also forests. Note that $F = F_1 \cup F_2$. Let $T_1' = V(F_1) \cap T$ and $T_2' = V(F_2) \cap T$. Since all the connected components in $V(F)$ contain at least one vertex from X and X_t is a bag in the tree decomposition, each connected component in F_1 as well as in F_2 contains at least one vertex from X. Let $C_1', \ldots, C_{q'}$ be the connected components of F_1 and $C_1'', \ldots, C_{q''}''$ be the connected components in F_2. Let $\mathcal{P}_1 = \{X \cap V(C_i'), \ldots, X \cap V(C_{q'}')\}$ and $\mathcal{P}_2 = \{X \cap V(C_i''), \ldots, X \cap V(C_{q''}'')\}$. Consider a new part C', from one of the partitions $\mathcal{P}_1$ or $\mathcal{P}_2$, and assume that $C' \subseteq C \in \mathcal{P}$. Let $r_{C'}$ be the unique vertex, in $V(C')$, that has minimum distance to the vertex $r_C \in C \cap X_{\mathsf{sp}}$. Then, each vertex in $V(C)$ has the shortness property with $r_{C'}$. This way, we obtain, for each $i \in [2]$, a set X_{sp}^i defined by taking exactly one vertex from each part of $\mathcal{P}_i$. Thus, $c[t_1,X,\mathcal{P}_1,X_{\mathsf{sp}}^1] \leq \mathsf{recdist}(F_1)$ and $c[t_2,X,\mathcal{P}_2,X_{\mathsf{sp}}^2] \leq \mathsf{recdist}(F_2)$. By induction hypothesis, $\mathcal{A}[\ell_{t_i},(X_{t_i},X,\mathcal{P}_i,X_{\mathsf{sp}}^i,T_i')] \leq c[t_i,X,\mathcal{P}_i,X_{\mathsf{sp}}^i]$ for $i \in \{1,2\}$. The definitions of F, F_1 and F_2 implies that $\mathcal{P} = \mathcal{P}_1 \sqcup \mathcal{P}_2$. Also, notice that the tuple $(\mathcal{P}_1,\mathcal{P}_2,X_{\mathsf{sp}}^1,X_{\mathsf{sp}}^2) \in \mathbb{Q}$. By Equations 9.11 and 9.12, it must be true that $\mathcal{A}[\ell_t,(X_t,X,\mathcal{P},X_{\mathsf{sp}},T')] \leq \mathcal{A}[\ell_t-1,(X_t,X,\mathcal{P}_1,X_{\mathsf{sp}}^1,T_1')] + \mathcal{A}[\ell_t-1,(X_t,X,\mathcal{P}_2,X_{\mathsf{sp}}^2,T_2')] \leq \mathsf{recdist}(F_1) + \mathsf{recdist}(F_2) = \mathsf{recdist}(F)$. This concludes the proof. □

Finally, we describe the subexponential algorithm for Rectilinear Steiner Arborescence.

Theorem 9.2. Rectilinear Steiner Arborescence *can be solved in time* $2^{\mathcal{O}(\sqrt{n}\log n)}n^{\mathcal{O}(1)}$.

Proof. We take as input a set T of n terminal points, the Hanan grid G of T and the weight function $\mathsf{recdist}$. Furthermore, $r \in T$is the root terminal. Then using Lemma 9.9 we compute a shortest path RSA $\widehat{S}$. By Lemma 9.12, we know that there is an optimal Steiner tree S_{opt} with $\mathbf{tw}(\widehat{S} \cup S_{\mathsf{opt}}) \leq 41\sqrt{n}$. Based on the shortest path RSA $\widehat{S}$, we apply Lemma 9.13, to enumerate all possible types D of G. We fix an integer $N = 3(|V(G)| + |E(G)|)$. For each $i \in [N]$ and each type D, the algorithm computes values $\mathcal{A}[i,D]$, according to Equations 9.7 and 9.12. The values in $\mathcal{A}[,]$ are filled in the increasing order of i. Finally, the algorithm outputs $\min_{i\in[N]} \mathcal{A}[i,(\{r\},\{r\},\{\{r\}\},\{r\},T)]$.

For the hypothetical set S, fix an optimal nice tree decomposition $\mathcal{T}$, rooted at node z. Add the root terminal r to each bag of the nice tree decomposition (as described in the proof of Lemma 9.15). The treewidth of this tree decomposition is bounded by $41\sqrt{n}+1$. Let t be a node in the tree decomposition, of level ℓ_t. Let X_t be the bag of t and V_t be the union of bags in the subtree rooted at t. Let $T' = T \cap V_t$. Suppose $X \subseteq X_t$, $\mathcal{P}$ is a partition of X, and the set X_{sp} is defined by selecting exactly one vertex from each part of $\mathcal{P}$. By definition, $c[t, X, \mathcal{P}, X_{\mathsf{sp}}]$ is the size of a locally-rooted subgraph of type $(X_t, X, \mathcal{P}, X_{\mathsf{sp}}, T')$. Then, Lemmata 9.14 and 9.15 imply that $\mathcal{A}[\ell_t, (X_t, X, \mathcal{P}, X_{\mathsf{sp}}, T')] = c[t, X, \mathcal{P}, X_{\mathsf{sp}}]$. In particular, for the root z of the tree decomposition, $\mathcal{A}[\ell_z, (\{r\}, \{r\}, \{\{r\}\}, \{r\}, T)] = c[z, \{r\}, \{\{r\}\}, \{r\}]$. Notice, that $c[z, \{r\}, \{\{r\}\}, \{r\}]$ is the size of a minimum rectilinear Steiner arborescence of G.

On the other hand, by Lemma 9.14, for all $i \in [N]$, if $A[i, (\{r\}, \{r\}, \{\{r\}\}, \{r\}, T)] = \ell$ then there is a locally-rooted subgraph F that connects all the terminals of T, and that satisfies $\mathsf{recdist}(F) \leq \ell$. As the algorithm outputs $\min_{i \in [N]} \mathcal{A}[i, (\{r\}, \{r\}, \{\{r\}\}, \{r\}, T)]$, it must output the weight of a minimum rectilinear Steiner arborescence of G.This proves the correctness of the algorithm.

The size of the table $\mathcal{A}[,]$ is $N \cdot 2^{\mathcal{O}(\sqrt{n}\log n)}$ and each entry can be filled in time $2^{\mathcal{O}(\sqrt{n}\log n)} n^{\mathcal{O}(1)}$. Thus, the running time of the algorithm is $2^{\mathcal{O}(\sqrt{n}\log n)} n^{\mathcal{O}(1)}$. Using standard back-tracking tricks we can also output an optimal RSA. This concludes the proof. □

9.5 Chapter Summary

We exhibit the first deterministic subexponential algorithms for Rectilinear Steiner Tree and Rectilinear Steiner Arborescence. The running time of both the algorithms is $2^{\mathcal{O}(\sqrt{n}\log n)} n^{\mathcal{O}(1)}$ time. Finding a lower bound for the running time of an algorithm and exhibiting an algorithm with optimal running time, for both the problems, remain open for investigation.

Part III

Conclusion and References

Chapter 10

Conclusion and Future Directions

Parameterized complexity has come a long way and now can be considered a field in its own right. It has a rich set of techniques and machinery, both for algorithm design and lower bound proofs. First and foremost, it is always interesting to classify a parameterized problem according to the W-hierarchy of parameterized complexity. On the positive side, there is always an endeavour to design algorithms with improved running times with respect to known algorithms, for a parameterized problem. Similarly, it is always interesting to try for better bounds on the size of a kernel for a parameterized problem. On the negative side, there are techniques to show lower bounds for the running time of an FPT algorithm, based on standard complexity theoretic assumptions. Lower bounds for the size of a kernel of a parameterized problem are also studied. In this thesis, problems were considered in the parameterized setting.

In graph partitioning, we studied Vertex (r, ℓ)-Partization and Edge (r, ℓ)-Partization on general graphs. We completed the dichotomy for Vertex (r, ℓ)-Partization and almost did the same for Edge (r, ℓ)-Partization. We also explored Vertex (r, ℓ)-Partization in perfect graphs, concluding that the problem is FPT but most likely with no polynomial kernel. From here, we generalised the notion of partitioning, and considered $(\mathcal{F}_1, \mathcal{F}_2)$-Partition, for two hereditary graph families $\mathcal{F}_1$ and $\mathcal{F}_2$. Here, the partitioning of the graph is actually a two-partitioning such that $G[V_1] \in \mathcal{F}_1$ while $G[V_2] \in \mathcal{F}_2$. The question is whether we can delete at most k vertices of the input graph to obtain a subgraph that has an $(\mathcal{F}_1, \mathcal{F}_2)$-Partition. We exhibit a technique to obtain FPT algorithms for a wide range of problems, Vertex (r, ℓ)-Partization being one. In fact, the technique uses a relation between $(\mathcal{F}_1, \mathcal{F}_2)$-Partition and the Two-Party model of Communication complexity for computing a binary function. This brings forward questions regarding parameterization in communication complexity. Lastly, we recall that Odd Cycle Transversal is a special case of Vertex (r, ℓ)-Partization. We study the dual problem of Even Cycle Transversal, and obtain the current fastest FPT algorithm. It should be mentioned that it is a randomized algorithm.

We also considered several geometric covering problems. Many of the problems are variants of Set Cover with the restriction that the set systems satisfy some geometric constraint. The set systems we have considered are lines, hyperplanes, unit squares, disks and sets of bounded intersection. We settled the parameterized complexity of most of the geometric covering problems we considered. We also explored the Conflict-Free Colouring and Unique-Maximum Colouring problems, which have important applications in computational geometry. In both the problems, the input is a set system, and the objective is to determine if the elements of the set system can be conflict-free coloured, or unique-maximum coloured, with a given number r of colours. We considered the problem of finding the maximum-sized subsystem, of the input set system, that can be conflict-free coloured, or unique-maximum coloured, with r colours. We obtained the current best exact algorithm for these two problems. We also addressed the parameterized version,

where the objective is to determine whether there is a subsystem with at least k sets, that can be conflict-free coloured, or unique- maximum coloured, with r colours. Here k is the parameter. We showed that both the problems are FPT. Finally, we considered the Rectilinear Steiner Tree and Rectilinear Steiner Arborescence problems, and gave the first subexponential time exact algorithms for determining minimum-sized rectilinear Steiner trees or rectilinear Steiner arborescences.

In most of the considered problems, the questions considered lead to many more open parameterized problems, opening up many directions of research in the future. At the end of each chapter, a description is given of some important open problems related to the problem considered in the chapter. They are some of the relevant problems that we hope to explore in future.

10.1 Related potential projects

Parameterized Communication Complexity: The notions of the two-party model for communication complexity has been described in Chapter 4. The Clique vs Independent Set problem is a cornerstone problem in this model of communication complexity. Lower bounds for the communication complexity of this problem are being seeked actively. Most of the existing lower bounds, except for one trivial bound, exhibit an infinite family of graphs for which the lower bound holds. This is where parameterization could be brought in. A parameterization can be thought of as a partitioning of all input instances. In some sense, a well-chosen parameter distinguishes between hard instances and easy instances. The concept of parameterized communication complexity could bring about a systematic study of nuanced lower bounds, where the lower bound has a dependence on the parameter associated with the input function.

Geometric Steiner Tree: Rectilinear Steiner Tree is in the intersection of geometric steiner tree variants and graphic steiner tree problems. This problem was originally defined in the geometric setting, but has an alternative graphic definition. In Chapter 9, we utilized the graphic properties to obtain efficient algorithms. It would be interesting to understand how the techniques can be adopted for other geometric variants, where our techniques break down and what adaptions can be made.

Art Gallery Problems: The Art Gallery Problem generates a lot of interest in the computational geometry community. Essentially, the problem takes as input a polygon, a set S of points in the polygon and another set S' of potential guard points from the polygon. The objective is to use as few guard points from S' as possible to cover all the points of S. There are several notions of a point p_1 covering another point p_2. The most popular one is when the line segment joining p_1 and p_2 is completely contained inside the input polygon. So far, there has been little interest in studying parameterized versions of the many variants of this problem. Given that most questions are hard in the classical sense, this would be a good area to bring in parameterized complexity. Not only are there algorithmic questions of finding optimal guard sets, but there is also hope of refining known combinatorial bounds when the input polygons are known to be associated with a certain parameter.

Bibliography

[Abu-Khzam 2010] Faisal N. Abu-Khzam. *A kernelization algorithm for d-Hitting Set.* J. Comput. Syst. Sci., vol. 76, no. 7, pages 524–531, 2010. 60

[Addario-Berry 2010] Louigi Addario-Berry, W.S. Kennedy, Andrew D. King, Zhentao Li and Bruce Reed. *Finding a maximum-weight induced -partite subgraph of an -triangulated graph.* Discrete Applied Mathematics, vol. 158, no. 7, pages 765 – 770, 2010. Third Workshop on Graph Classes, Optimization, and Width Parameters Eugene, Oregon, USA, October 2007. 51

[Agarwal 2005] Amit Agarwal, Moses Charikar, Konstantin Makarychev and Yury Makarychev. $O(\sqrt{\log n})$ *approximation algorithms for min UnCut, min 2CNF deletion, and directed cut problems.* In STOC, pages 573–581, 2005. 35, 40, 41

[Ajwani 2012] Deepak Ajwani, Khaled Elbassioni, Sathish Govindarajan and Saurabh Ray. *Conflict-free coloring for rectangle ranges using O (n. 382) colors.* Discrete & Computational Geometry, vol. 48, no. 1, pages 39–52, 2012. 159

[Alber 2004] Jochen Alber and Ji Fiala. *Geometric separation and exact solutions for the parameterized independent set problem on disk graphs.* Journal of Algorithms, vol. 52, no. 2, pages 134 – 151, 2004. 25

[Alon 1995] Noga Alon, Raphael Yuster and Uri Zwick. *Color-coding.* J. ACM, vol. 42, no. 4, pages 844–856, July 1995. 161, 165, 169, 171, 173

[Amano 2014] Kazuyuki Amano. *Some improved bounds on communication complexity via new decomposition of cliques.* Discrete Applied Mathematics, vol. 166, pages 249–254, 2014. 61

[Appel 1977] K. Appel and W. Haken. *Every planar map is four colorable. Part I: Discharging.* Illinois J. Math., vol. 21, no. 3, pages 429–490, 09 1977. 20

[Ashok 2015a] Pradeesha Ashok, Aditi Dudeja and Sudeshna Kolay. *Exact and FPT Algorithms for Max-Conflict Free Coloring in Hypergraphs.* In Algorithms and Computation - 26th International Symposium, ISAAC 2015, Nagoya, Japan, December 9-11, 2015, Proceedings, pages 271–282, 2015. 160

[Ashok 2015b] Pradeesha Ashok, Sudeshna Kolay, Neeldhara Misra and Saket Saurabh. *Unique Covering Problems with Geometric Sets.* In Computing and Combinatorics - 21st International Conference, COCOON 2015, Beijing, China, August 4-6, 2015, Proceedings, pages 548–558, 2015. 144

[Ashok 2016] Pradeesha Ashok, Sudeshna Kolay and Saket Saurabh. *Parameterized Complexity of Red Blue Set Cover for Lines.* In LATIN 2016: Theoretical Informatics - 12th Latin American Symposium, Ensenada, Mexico, April 11-15, 2016, Proceedings, pages 96–109, 2016. 113

[Baste 2015] Julien Baste, Luerbio Faria, Sulamita Klein and Ignasi Sau. *Parameterized complexity dichotomy for (r, l)-Vertex Deletion.* CoRR, vol. abs/1504.05515, 2015. 34, 89

[Becker 2000] Ann Becker, Reuven Bar-Yehuda and Dan Geiger. *Randomized Algorithms for the Loop Cutset Problem.* (JAIR), vol. 12, pages 219–234, 2000. 92

[Binkele-Raible 2012] Daniel Binkele-Raible, Henning Fernau, Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh and Yngve Villanger. *Kernel(s) for problems with no kernel: On out-trees with many leaves.* ACM Trans. Algorithms, vol. 8, no. 4, page 38, 2012. 18

[Björklund 2007] Andreas Björklund, Thore Husfeldt, Petteri Kaski and Mikko Koivisto. *Fourier meets möbius: fast subset convolution.* In Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007, pages 67–74, 2007. 180

[Bodlaender 2009] Hans L. Bodlaender, Fedor V. Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh and Dimitrios M. Thilikos. *(Meta) Kernelization.* In FOCS 2009, pages 629–638, 2009. 92

[Bodlaender 2011] Hans L. Bodlaender, Stéphan Thomassé and Anders Yeo. *Kernel bounds for disjoint cycles and disjoint paths.* Theor. Comput. Sci., vol. 412, no. 35, pages 4570–4578, 2011. 19

[Bodlaender 2013] Hans L. Bodlaender, Bart M. P. Jansen and Stefan Kratsch. *Preprocessing for Treewidth: A Combinatorial Analysis through Kernelization.* SIAM J. Discrete Math., vol. 27, no. 4, pages 2108–2142, 2013. 24

[Bodlaender 2014] Hans L. Bodlaender, Bart M. P. Jansen and Stefan Kratsch. *Kernelization Lower Bounds by Cross-Composition.* SIAM J. Discrete Math., vol. 28, no. 1, pages 277–305, 2014. 24

[Bousquet 2014] Nicolas Bousquet, Aurélie Lagoutte and Stéphan Thomassé. *Clique versus independent set.* Eur. J. Comb., vol. 40, pages 73–92, 2014. 63

[Brandstdt 1998] Andreas Brandstdt, Van Bang Le and Thomas Szymczak. *The complexity of some problems related to Graph 3-colorability.* Discrete Applied Mathematics, vol. 89, no. 13, pages 59 – 73, 1998. 33, 36

[Brazil 2015] Marcus Brazil and Martin Zachariasen. Optimal interconnection trees in the plane: Theory, algorithms and applications. Springer Publishing Company, Incorporated, 2015. 179, 180

[Broersma 2002] Hajo Broersma, Fedor V. Fomin, Jaroslav Nesetril and Gerhard J. Woeginger. *More About Subcolorings.* Computing, vol. 69, no. 3, pages 187–203, 2002. 20

[Byskov 2004] J.M. Byskov. *Exact algorithms for graph colouring and exact satisfiability.* 2004. 87

[Cairnie 1998] Niall Cairnie and Keith Edwards. *The achromatic number of bounded degree trees.* Discrete Mathematics, vol. 188, no. 13, pages 87 – 97, 1998. 20

[Cao 2015a] Yixin Cao. *Unit Interval Editing is Fixed-Parameter Tractable.* In ICALP 2015, volume 9134 of *LNCS*, pages 306–317. Springer, 2015. 91

[Cao 2015b] Yixin Cao and Dániel Marx. *Interval Deletion Is Fixed-Parameter Tractable.* ACM Transactions on Algorithms, vol. 11, no. 3, pages 21:1–21:35, 2015. 91

[Carr 2000] Robert D Carr, Srinivas Doddi, Goran Konjevod and Madhav V Marathe. *On the red-blue set cover problem.* In SODA, volume 9, pages 345–353, 2000. 111, 113

[Chambers 2008] Erin W. Chambers, ric Colin de Verdire, Jeff Erickson, Francis Lazarus and Kim Whittlesey. *Splitting (complicated) surfaces is hard.* Computational Geometry, vol. 41, no. 12, pages 94 – 110, 2008. Special Issue on the 22nd European Workshop on Computational Geometry (EuroCG)22nd European Workshop on Computational Geometry. 24

[Chan 2015] Timothy M Chan and Nan Hu. *Geometric red–blue set cover for unit squares and related problems.* Computational Geometry, vol. 48, no. 5, pages 380–385, 2015. 113

[Chan 2016] Timothy M. Chan. *Improved Deterministic Algorithms for Linear Programming in Low Dimensions.* In Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016, pages 1213–1219, 2016. 24

[Chazelle 1996] Bernard Chazelle and Ji Matouek. *On Linear-Time Deterministic Algorithms for Optimization Problems in Fixed Dimension.* Journal of Algorithms, vol. 21, no. 3, pages 579 – 597, 1996. 24

[Cheilaris 2011] Panagiotis Cheilaris and Géza Tóth. *Graph unique-maximum and conflict-free colorings.* Journal of Discrete Algorithms, vol. 9, no. 3, pages 241–251, 2011. 160

[Chen 2010] Jianer Chen, Iyad A. Kanj and Ge Xia. *Improved upper bounds for vertex cover.* Theor. Comput. Sci., vol. 411, no. 40-42, pages 3736–3756, 2010. 21, 90

[Chudnovsky 2006] Maria Chudnovsky, Neil Robertson, Paul D. Seymour and Robin Thomas. *The strong perfect graph theorem.* Annals of Mathematics, vol. 164, pages 51–229, 2006. 52

[Cook 1971] Stephen A. Cook. *The Complexity of Theorem-proving Procedures.* In STOC, pages 151–158, New York, NY, USA, 1971. ACM. 43

[Cormen 2009] Thomas H Cormen. Introduction to algorithms. MIT press, 2009. 17

[Courcelle 1990] Bruno Courcelle. *The Monadic Second-Order Logic of Graphs. I. Recognizable Sets of Finite Graphs.* Inf. Comput., vol. 85, no. 1, pages 12–75, 1990. 48

[Cygan 2013] Marek Cygan and Marcin Pilipczuk. *Split Vertex Deletion meets Vertex Cover: New fixed-parameter and exact exponential-time algorithms.* Inf. Process. Lett., vol. 113, no. 5-6, pages 179–182, 2013. 21, 63, 64, 77, 79

[Cygan 2015] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk and Saket Saurabh. Parameterized algorithms. Springer, 2015. 19, 46, 88, 89, 181, 183, 195, 197

[Cygan 2016] Marek Cygan, Fedor V. Fomin, Alexander Golovnev, Alexander S. Kulikov, Ivan Mihajlin, Jakub Pachocki and Arkadiusz Socala. *Tight Bounds for Graph Homomorphism and Subgraph Isomorphism.* In Proceedings of the Twenty-Seventh

Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016, pages 1643–1649, 2016. 21

[Dailey 1980] David P. Dailey. *Uniqueness of colorability and colorability of planar 4-regular graphs are NP-complete.* Discrete Mathematics, vol. 30, no. 3, pages 289 – 293, 1980. 20

[de Figueiredo 2000] Celina M.H. de Figueiredo, Sulamita Klein, Yoshiharu Kohayakawa and Bruce A. Reed. *Finding Skew Partitions Efficiently.* Journal of Algorithms, vol. 37, no. 2, pages 505 – 521, 2000. 20

[Deineko 2004] Vladimir G. Deineko, Michael Hoffmann, Yoshio Okamoto and Gerhard J. Woeginger. Computing and combinatorics: 10th annual international conference, cocoon 2004, jeju island, korea, august 17-20, 2004. proceedings, chapitre The Traveling Salesman Problem with Few Inner Points, pages 268–277. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. 25

[Demaine 2005] Erik D. Demaine, Fedor V. Fomin, Mohammad Taghi Hajiaghayi and Dimitrios M. Thilikos. *Subexponential parameterized algorithms on bounded-genus graphs and* H*-minor-free graphs.* J. ACM, vol. 52, no. 6, pages 866–893, 2005. 112

[Demaine 2008] Erik D Demaine, Uriel Feige, MohammadTaghi Hajiaghayi and Mohammad R Salavatipour. *Combination Can Be Hard: Approximability of the Unique Coverage Problem.* SIAM J. Comput. (), vol. 38, no. 4, pages 1464–1483, 2008. 143

[Deneen 1994] L. Deneen, G. Shute and C. Thomborson. *A probably fast, provably optimal algorithm for rectilinear Steiner trees.* Random Structures Algorithms, vol. 5, no. 4, pages 535–557, 1994. 180

[Diestel 2012] Reinhard Diestel. Graph theory, 4th edition, volume 173 of *Graduate texts in mathematics.* Springer, 2012. 15, 93, 94

[Dom 2008] Michael Dom, Jiong Guo, Rolf Niedermeier and Sebastian Wernicke. *Red-blue covering problems and the consecutive ones property.* J. Discrete Algorithms, vol. 6, no. 3, pages 393–407, 2008. 113

[Dom 2014] Michael Dom, Daniel Lokshtanov and Saket Saurabh. *Kernelization Lower Bounds Through Colors and IDs.* ACM Trans. Algorithms, vol. 11, no. 2, pages 13:1–13:20, 2014. 117, 151

[Downey 2012] Rodney G Downey and Michael Ralph Fellows. Parameterized complexity. Springer Science & Business Media, 2012. 19, 112, 117, 146

[Dreyfus 1971] S. E. Dreyfus and R. A. Wagner. *The steiner problem in graphs.* Networks, vol. 1, no. 3, pages 195–207, 1971. 180

[Dyer] M. E. Dyer and A. M. Frieze. *A randomized algorithm for fixed-dimensional linear programming.* Mathematical Programming, vol. 44, no. 1, pages 203–212. 24

[Edelsbrunner 1986] H. Edelsbrunner and R. Waupotitsch. *Computing a ham-sandwich cut in two dimensions.* Journal of Symbolic Computation, vol. 2, no. 2, pages 171 – 178, 1986. 25

[Eppstein 2010] David Eppstein, Maarten Löffler and Darren Strash. *Listing All Maximal Cliques in Sparse Graphs in Near-Optimal Time*. In Algorithms and Computation - 21st International Symposium, ISAAC 2010, Jeju Island, Korea, December 15-17, 2010, Proceedings, Part I, pages 403–414, 2010. 46

[Even 2002] G. Even, Z. Lotker, D. Ron and S. Smorodinsky. *Conflict-free colorings of simple geometric regions with applications to frequency assignment in cellular networks*. In Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on, pages 691–700, 2002. 159

[Feder 1999] Tomas Feder, Pavol Hell, Sulamita Klein and Rajeev Motwani. *Complexity of Graph Partition Problems*. In Proceedings of the Thirty-first Annual ACM Symposium on Theory of Computing, STOC '99, pages 464–472, New York, NY, USA, 1999. ACM. 62

[Feder 2003] Tomas Feder, Pavol Hell, Sulamita Klein and Rajeev Motwani. *List partitions*. SIAM Journal on Discrete Mathematics, vol. 16, no. 3, pages 449–478, 2003. 20, 33, 36

[Feder 2011] Tomás Feder, Pavol Hell and Shekoofeh Nekooei Rizi. *Partitioning Chordal Graphs*. Electronic Notes in Discrete Mathematics, vol. 38, pages 325–330, 2011. 33, 51

[Fellows 2008] Michael R. Fellows, Christian Knauer, Naomi Nishimura, Prabhakar Ragde, Frances A. Rosamond, Ulrike Stege, Dimitrios M. Thilikos and Sue Whitesides. *Faster Fixed-Parameter Tractable Algorithms for Matching and Packing Problems*. Algorithmica, vol. 52, no. 2, pages 167–176, 2008. 112

[Fellows 2009] Michael R. Fellows, Danny Hermelin, Frances Rosamond and Stéphane Vialette. *On the Parameterized Complexity of Multiple-interval Graph Problems*. Theor. Comput. Sci., vol. 410, no. 1, pages 53–61, January 2009. 152

[Fiorini 2010] Samuel Fiorini, Gwenaël Joret and Ugo Pietropaoli. *Hitting Diamonds and Growing Cacti*. In IPCO 2010, pages 191–204, 2010. 92, 94, 106

[Flum 2006] J. Flum and M. Grohe. Parameterized complexity theory (texts in theoretical computer science. an eatcs series). Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. 18, 19

[Fomin 2008] Fedor V. Fomin, Serge Gaspers, Artem V. Pyatkin and Igor Razgon. *On the Minimum Feedback Vertex Set Problem: Exact and Enumeration Algorithms*. Algorithmica, vol. 52, no. 2, pages 293–307, 2008. 88

[Fomin 2009] Fedor V. Fomin, Serge Gaspers, Saket Saurabh and Alexey A. Stepanov. *On Two Techniques of Combining Branching and Treewidth*. Algorithmica, vol. 54, no. 2, pages 181–207, 2009. 112

[Fomin 2010] Fedor V Fomin and Dieter Kratsch. Exact exponential algorithms. Springer, 2010. 161, 162

[Fomin 2011] Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, Geevarghese Philip and Saket Saurabh. *Hitting forbidden minors: Approximation and Kernelization*. In Thomas Schwentick and Christoph Dürr, editeurs, STACS 2011, volume 9 of *(LIPIcs)*, pages 189–200, Dagstuhl, Germany, 2011. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. 92

[Fomin 2012] Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra and Saket Saurabh. *Planar F-Deletion: Approximation, Kernelization and Optimal FPT Algorithms.* In FOCS 2012, pages 470–479, 2012. 91, 92

[Fomin 2013] Fedor V. Fomin and Yngve Villanger. *Subexponential Parameterized Algorithm for Minimum Fill-In.* SIAM J. Comput., vol. 42, no. 6, pages 2197–2216, 2013. 91

[Fomin 2014a] Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan and Saket Saurabh. *Representative Sets of Product Families.* In ESA, volume 8737, pages 443–454, 2014. 139

[Fomin 2014b] Fedor V. Fomin, Daniel Lokshtanov and Saket Saurabh. *Efficient Computation of Representative Sets with Applications in Parameterized and Exact Algorithms.* In SODA, pages 142–151, 2014. 139

[Fomin 2016] Fedor V. Fomin, Sudeshna Kolay, Daniel Lokshtanov, Fahad Panolan and Saket Saurabh. *Subexponential Algorithms for Rectilinear Steiner Tree and Arborescence Problems.* In 32nd International Symposium on Computational Geometry, SoCG 2016, June 14-18, 2016, Boston, MA, USA, pages 39:1–39:15, 2016. 182

[Fortnow 2011] Lance Fortnow and Rahul Santhanam. *Infeasibility of instance compression and succinct PCPs for NP.* J. Comput. Syst. Sci., vol. 77, no. 1, pages 91–106, 2011. 19, 57

[Fuchs 2006] B. Fuchs, W. Kern, D. Mlle, S. Richter, P. Rossmanith and X. Wang. *Dynamic programming for minimum Steiner trees.* Theory Comput Syst, pages 270–280, 2006. 180

[Ganley 1997] Joseph L. Ganley and James P. Cohoon. *Improved Computation of Optimal Rectilinear Steiner Minimal Trees.* Int. J. Comput. Geometry Appl., vol. 7, no. 5, pages 457–472, 1997. 180

[Ganley 1999] Joseph L Ganley. *Computing optimal rectilinear Steiner trees: A survey and experimental evaluation.* Discrete Applied Mathematics, vol. 90, no. 13, pages 161 – 171, 1999. 180

[Garey 1977] M. R. Garey and David S. Johnson. *The Rectilinear Steiner Tree Problem in NP Complete.* SIAM Journal of Applied Mathematics, vol. 32, pages 826–834, 1977. 180

[Garey 2002] Michael R Garey and David S Johnson. Computers and intractability, volume 29. wh freeman New York, 2002. 20, 26, 33, 51

[Gargano 2015] Luisa Gargano and Adele A. Rescigno. *Complexity of Conflict-Free Colourings of Graphs.* Theoretical Computer Science, vol. 566, pages 39–49, 2015. 159, 160, 161

[Ghosh 2015] Esha Ghosh, Sudeshna Kolay, Mrinal Kumar, Pranabendu Misra, Fahad Panolan, Ashutosh Rai and M. S. Ramanujan. *Faster Parameterized Algorithms for Deletion to Split Graphs.* Algorithmica, vol. 71, no. 4, pages 989–1006, 2015. 21, 22

[Giannopoulos 2009] Panos Giannopoulos, Christian Knauer and Günter Rote. *The parameterized complexity of some geometric problems in unbounded dimension.* In Parameterized and Exact Computation, pages 198–209. Springer, 2009. 25

[Giannopoulou 2015] Archontia C. Giannopoulou, Bart M. P. Jansen, Daniel Lokshtanov and Saket Saurabh. *Uniform Kernelization Complexity of Hitting Forbidden Minors.* In ICALP 2015, volume 9134 of *LNCS*, pages 629–641. Springer, 2015. 91

[Golumbic 2004] Martin Charles Golumbic. Algorithmic graph theory and perfect graphs (annals of discrete mathematics, vol 57). North-Holland Publishing Co., Amsterdam, The Netherlands, The Netherlands, 2004. 20, 22

[Göös 2015a] Mika Göös. *Lower Bounds for Clique vs. Independent Set.* In IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015, pages 1066–1076, 2015. 61, 63

[Göös 2015b] Mika Göös, T. S. Jayram, Toniann Pitassi and Thomas Watson. *Randomized Communication vs. Partition Number.* Electronic Colloquium on Computational Complexity (ECCC), vol. 22, page 169, 2015. 61

[Göös 2015c] Mika Göös, Toniann Pitassi and Thomas Watson. *Deterministic Communication vs. Partition Number.* In IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015, pages 1077–1088, 2015. 61

[Graham 1972] Ronald L. Graham. *An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set.* Inf. Process. Lett., vol. 1, no. 4, pages 132–133, 1972. 24

[Gu 2012] Qian-Ping Gu and Hisao Tamaki. *Improved Bounds on the Planar Branchwidth with Respect to the Largest Grid Minor Size.* Algorithmica, vol. 64, no. 3, pages 416–453, 2012. 187

[Guo 2006] Jiong Guo, Jens Gramm, Falk Hüffner, Rolf Niedermeier and Sebastian Wernicke. *Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization.* J. Comput. Syst. Sci., vol. 72, no. 8, pages 1386–1396, 2006. 21, 47

[Gupta 2012] Sushmita Gupta, Venkatesh Raman and Saket Saurabh. *Maximum r-Regular Induced Subgraph Problem: Fast Exponential Algorithms and Combinatorial Bounds.* SIAM J. Discrete Math., vol. 26, no. 4, pages 1758–1780, 2012. 88

[Gyárfás 1998] András Gyárfás. *Generalized Split Graphs and Ramsey Numbers.* J. Comb. Theory, Ser. A, vol. 81, no. 2, pages 255–261, 1998. 35, 40, 52

[Hanan 1966] M. Hanan. *On Steiner's problem with rectilinear distance.* SIAM Journal on Applied Mathematics, no. 14, pages 255–265, 1966. 179

[Har-Peled 2005] Sariel Har-Peled and Shakhar Smorodinsky. *Conflict-free coloring of points and simple regions in the plane.* Discrete & Computational Geometry, vol. 34, no. 1, pages 47–70, 2005. 159

[Heggernes 2013] Pinar Heggernes, Dieter Kratsch, Daniel Lokshtanov, Venkatesh Raman and Saket Saurabh. *Fixed-parameter algorithms for Cochromatic Number and Disjoint Rectangle Stabbing via iterative localization.* Inf. Comput., vol. 231, pages 109–116, 2013. 21, 26, 51, 52, 53, 54, 56

[Hell 1976] Pavol Hell and Donald J. Miller. *Graph with given achromatic number.* Discrete Mathematics, vol. 16, no. 3, pages 195 – 207, 1976. 21

[Hochbaum 1987] Dorit S Hochbaum and Wolfgang Maass. *Fast approximation algorithms for a nonconvex covering problem.* Journal of algorithms, vol. 8, no. 3, pages 305–323, 1987. 144

[Huang 2012] Hao Huang and Benny Sudakov. *A counterexample to the Alon-Saks-Seymour conjecture and related problems.* Combinatorica, vol. 32, no. 2, pages 205–219, 2012. 61

[Hüffner 2009] Falk Hüffner. *Algorithm Engineering for Optimal Graph Bipartization.* J. Graph Algorithms Appl., vol. 13, no. 2, pages 77–98, 2009. 21

[Hwang 1976] F. K. Hwang. *On Steiner minimal trees with rectilinear distance.* SIAM Journal on Applied Mathematics, no. 30, pages 104–114, 1976. 181

[Hwang 1992] F. K. Hwang, D. S. Richards and P. Winter. *The Steiner tree problem.* Annals of Discrete Mathematics, vol. 53, 1992. 179, 180

[Impagliazzo 2001] Russell Impagliazzo, Ramamohan Paturi and Francis Zane. *Which Problems Have Strongly Exponential Complexity?* J. Comput. Syst. Sci., vol. 63, no. 4, pages 512–530, 2001. 115

[Iwata 2014] Yoichi Iwata, Keigo Oka and Yuichi Yoshida. *Linear-Time FPT Algorithms via Network Flow.* In SODA, pages 1749–1761, 2014. 21

[Jansen 2013] Bart M. P. Jansen and Hans L. Bodlaender. *Vertex Cover Kernelization Revisited - Upper and Lower Bounds for a Refined Parameter.* Theory Comput. Syst., vol. 53, no. 2, pages 263–299, 2013. 24

[Jansen 2014] Bart M. P. Jansen, Daniel Lokshtanov and Saket Saurabh. *A Near-Optimal Planarization Algorithm.* In Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014, pages 1802–1811, 2014. 90

[Joret 2014] Gwenaël Joret, Christophe Paul, Ignasi Sau, Saket Saurabh and Stéphan Thomassé. *Hitting and Harvesting Pumpkins.* SIAM J. Discrete Math., vol. 28, no. 3, pages 1363–1390, 2014. 91, 92

[Jukna 2011] Stasys Jukna. Extremal combinatorics: with applications in computer science. Springer Science & Business Media, 2011. 164

[Kára 2006] Jan Kára and Jan Kratochvíl. Parameterized and exact computation: Second international workshop, iwpec 2006, zürich, switzerland, september 13-15, 2006. proceedings, chapitre Fixed Parameter Tractability of Independent Set in Segment Intersection Graphs, pages 166–174. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. 25

[Karp 1972] Richard M. Karp. *Reducibility Among Combinatorial Problems.* In Proceedings of a symposium on the Complexity of Computer Computations, pages 85–103, 1972. 111, 143

[Kim 2015] Eun Jung Kim, Alexander Langer, Christophe Paul, Felix Reidl, Peter Rossmanith, Ignasi Sau and Somnath Sikdar. *Linear kernels and single-exponential algorithms via protrusion decompositions.* ACM Transactions on Algorithms (TALG), vol. 12, no. 2, page 21, 2015. 91, 92

[Kirkpatrick 1986] David G. Kirkpatrick and Raimund Seidel. *The Ultimate Planar Convex Hull Algorithm?* SIAM J. Comput., vol. 15, no. 1, pages 287–299, 1986. 24

[Klein 1996] Sulamita Klein and Celina M. H. De Figueiredo. *The NP-completeness of multi-partite cutset testing.* Congr. Numer, vol. 119, pages 217–222, 1996. 20

[Klein 2014] Philip N. Klein and Dániel Marx. *A subexponential parameterized algorithm for Subset TSP on planar graphs.* In Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014, pages 1812–1830, 2014. 181

[Knauer 2006] Christian Knauer and Andreas Spillner. Graph-theoretic concepts in computer science: 32nd international workshop, wg 2006, bergen, norway, june 22-24, 2006 revised papers, chapitre A Fixed-Parameter Algorithm for the Minimum Weight Triangulation Problem Based on Small Graph Separators, pages 49–57. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. 24

[Knauer 2011] Christian Knauer, Hans Raj Tiwary and Daniel Werner. *On the computational complexity of Ham-Sandwich cuts, Helly sets, and related problems.* In 28th International Symposium on Theoretical Aspects of Computer Science, STACS 2011, March 10-12, 2011, Dortmund, Germany, pages 649–660, 2011. 25

[Kociumaka 2014] Tomasz Kociumaka and Marcin Pilipczuk. *Faster deterministic Feedback Vertex Set.* Inf. Process. Lett., vol. 114, no. 10, pages 556–560, 2014. 90

[Kolay 2015a] Sudeshna Kolay, Daniel Lokshtanov, Fahad Panolan and Saket Saurabh. *Quick but Odd Growth of Cacti.* In 10th International Symposium on Parameterized and Exact Computation, IPEC 2015, September 16-18, 2015, Patras, Greece, pages 258–269, 2015. 92

[Kolay 2015b] Sudeshna Kolay and Fahad Panolan. *Parameterized Algorithms for Deletion to (r, ell)-Graphs.* In 35th IARCS Annual Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2015, December 16-18, 2015, Bangalore, India, pages 420–433, 2015. 34, 89

[Kratsch 2012] Stefan Kratsch and Magnus Wahlström. *Representative Sets and Irrelevant Vertices: New Tools for Kernelization.* In FOCS, pages 450–459, 2012. 42, 43

[Kratsch 2014a] Stefan Kratsch, Geevarghese Philip and Saurabh Ray. *Point Line Cover: The Easy Kernel is Essentially Tight.* In SODA, pages 1596–1606, 2014. 112

[Kratsch 2014b] Stefan Kratsch and Magnus Wahlström. *Compression via Matroids: A Randomized Polynomial Kernel for Odd Cycle Transversal.* ACM Transactions on Algorithms, vol. 10, no. 4, page 20, 2014. 21, 35, 40

[Krithika 2013] R. Krithika and N. S. Narayanaswamy. *Parameterized Algorithms for (r, l)-Partization.* J. Graph Algorithms Appl., vol. 17, no. 2, pages 129–146, 2013. 21, 51

[Kushilevitz 1997] Eyal Kushilevitz and Noam Nisan. Communication complexity. Cambridge University Press, New York, NY, USA, 1997. 65, 66

[Kzdy 1996] Andr E. Kzdy, Hunter S. Snevily and Chi Wang. *Partitioning permutations into increasing and decreasing subsequences.* Journal of Combinatorial Theory, Series A, vol. 73, no. 2, pages 353 – 359, 1996. 52, 60

[Langerman 2005] Stefan Langerman and Pat Morin. *Covering things with things.* Discrete & Computational Geometry, vol. 33, no. 4, pages 717–729, 2005. 25, 26, 112, 115, 116, 118, 146, 149, 150

[Lewis 1980] John M. Lewis and Mihalis Yannakakis. *The Node-Deletion Problem for Hereditary Properties is NP-Complete.* J. Comput. Syst. Sci., vol. 20, no. 2, pages 219–230, 1980. 91

[Lokshtanov 2009] Daniel Lokshtanov, Saket Saurabh and Somnath Sikdar. *Simpler Parameterized Algorithm for OCT.* In IWOCA, pages 380–384, 2009. 21

[Lokshtanov 2012] Daniel Lokshtanov and M. S. Ramanujan. *Parameterized Tractability of Multiway Cut with Parity Constraints.* In ICALP 2012, pages 750–761, 2012. 100

[Lokshtanov 2014] Daniel Lokshtanov, N. S. Narayanaswamy, Venkatesh Raman, M. S. Ramanujan and Saket Saurabh. *Faster Parameterized Algorithms Using Linear Programming.* ACM Transactions on Algorithms, vol. 11, no. 2, page 15, 2014. 21, 37, 90

[Lovász 1972] László Lovász. *A characterization of perfect graphs.* J. Comb. Theory, Ser. B, vol. 13, no. 2, pages 95–98, 1972. 52

[Lovász 1990] László Lovász. *Communication complexity: a survey.* Paths, flows, and VLSI-layout, pages 235–265, 1990. 61, 63, 66, 77

[Lovász 1994] László Lovász. *Stable sets and polynomials.* Discrete Mathematics, vol. 124, no. 1-3, pages 137–153, 1994. 63

[MacGillivray 1999] Gary MacGillivray and Min-Li Yu. *Generalized partitions of graphs.* Discrete Applied Mathematics, vol. 91, no. 13, pages 143 – 153, 1999. 20

[Marx 2005] Dániel Marx. *Efficient approximation schemes for geometric problems?* In ESA, pages 448–459. Springer, 2005. 25, 26, 112, 144

[Marx 2006] Dániel Marx. *Parameterized complexity of independence and domination on geometric graphs.* In Parameterized and Exact Computation, pages 154–165. Springer, 2006. 25, 128

[Marx 2013] Dániel Marx, Barry O'Sullivan and Igor Razgon. *Finding small separators in linear time via treewidth reduction.* ACM Transactions on Algorithms, vol. 9, no. 4, page 30, 2013. 35, 48

[Mathieson 2008] Luke Mathieson and Stefan Szeider. *The Parameterized Complexity of Regular Subgraph Problems and Generalizations.* In CATS, volume 77, pages 79–86, 2008. 120

[Matoušek] J. Matoušek, M. Sharir and E. Welzl. *A subexponential bound for linear programming.* Algorithmica, vol. 16, no. 4, pages 498–516. 24

[Matoušek 2002] Jiří Matoušek. Lectures on discrete geometry, volume 108. Springer New York, 2002. 145

[Megiddo 1982] Nimrod Megiddo and Arie Tamir. *On the Complexity of Locating Linear Facilities in the Plane.* Oper. Res. Lett., vol. 1, no. 5, pages 194–197, November 1982. 154

[Megiddo 1984] Nimrod Megiddo. *Linear Programming in Linear Time When the Dimension Is Fixed.* J. ACM, vol. 31, no. 1, pages 114–127, 1984. 24

[Misra 2012] Pranabendu Misra, Venkatesh Raman, M. S. Ramanujan and Saket Saurabh. *Parameterized Algorithms for Even Cycle Transversal.* In WG 2012, pages 172–183, 2012. 92

[Misra 2013] Neeldhara Misra, Hannes Moser, Venkatesh Raman, Saket Saurabh and Somnath Sikdar. *The Parameterized Complexity of Unique Coverage and Its Variants.* Algorithmica, pages 517–544, 2013. 146, 151, 152, 161, 163

[Moon] J. W. Moon and L. Moser. *On cliques in graphs.* Israel Journal of Mathematics, vol. 3, no. 1, pages 23–28. 87, 88

[Mulzer 2008] Wolfgang Mulzer and Günter Rote. *Minimum-weight triangulation is NP-hard.* Journal of the ACM (JACM), vol. 55, no. 2, page 11, 2008. 24

[Mustafa 2009] Nabil Mustafa and Saurabh Ray. *PTAS for geometric hitting set problems via local search.* In Proceedings of the 25th annual symposium on Computational geometry, pages 17–22. ACM, 2009. 144

[Naor 1995] Moni Naor, Leonard J. Schulman and Aravind Srinivasan. *Splitters and Near-Optimal Derandomization.* In 36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, 23-25 October 1995, pages 182–191, 1995. 169

[Nastansky] L. Nastansky, S. M. Selkow and N. F. Stewart. *Cost-minimal trees in directed acyclic graphs.* Zeitschrift für Operations Research, vol. 18, no. 1, pages 59–67. 181

[Nederlof 2013] Jesper Nederlof. *Fast Polynomial-Space Algorithms Using Inclusion-Exclusion.* Algorithmica, vol. 65, no. 4, pages 868–884, 2013. 180

[Niedermeier 2003] Rolf Niedermeier and Peter Rossmanith. *An efficient fixed-parameter algorithm for 3-Hitting Set.* Journal of Discrete Algorithms, vol. 1, no. 1, pages 89 – 102, 2003. Combinatorial Algorithms. 90

[Pach 2009] János Pach and Gábor Tardos. *Conflict-free colourings of graphs and hypergraphs.* Combinatorics, Probability and Computing, vol. 18, no. 05, pages 819–834, 2009. 27, 159

[Panolan 2015] Fahad Panolan, Geevarghese Philip and Saket Saurabh. *B-Chromatic Number: Beyond NP-Hardness.* In 10th International Symposium on Parameterized and Exact Computation, IPEC 2015, September 16-18, 2015, Patras, Greece, pages 389–401, 2015. 21

[Peleg 2007] David Peleg. *Approximation algorithms for the Label-Cover$_{MAX}$ and Red-Blue Set Cover problems.* J. Discrete Algorithms, vol. 5, no. 1, pages 55–64, 2007. 113

[Pilipczuk 2012] Marcin Pilipczuk and Michal Pilipczuk. *Finding a Maximum Induced Degenerate Subgraph Faster Than 2 n.* In Parameterized and Exact Computation - 7th International Symposium, IPEC 2012, Ljubljana, Slovenia, September 12-14, 2012. Proceedings, pages 3–12, 2012. 87

[Pilipczuk 2013] Marcin Pilipczuk, Michał Pilipczuk, Piotr Sankowski and Erik Jan van Leeuwen. *Subexponential-Time Parameterized Algorithm for Steiner Tree on Planar Graphs.* volume 20, pages 353–364, Dagstuhl, Germany, 2013. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. 180

[Pilipczuk 2014] Marcin Pilipczuk, Michal Pilipczuk, Piotr Sankowski and Erik Jan van Leeuwen. *Network Sparsification for Steiner Problems on Planar and Bounded-Genus Graphs.* In 55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014, pages 276–285, 2014. 180

[Pilipczuk 2015] Marcin Pilipczuk, Michal Pilipczuk and Marcin Wrochna. *Edge Bipartization faster than 2^k.* CoRR, vol. abs/1507.02168, 2015. 21

[Prömel 2002] H. J. Prömel and A. Steger. The Steiner tree problem. Advanced Lectures in Mathematics, Friedr. Vieweg & Sohn, Braunschweig, 2002. 179

[Raman 2008] Venkatesh Raman and Saket Saurabh. *Short cycles make W-hard problems hard: FPT algorithms for W-hard problems in graphs with no short cycles.* Algorithmica, vol. 52, no. 2, pages 203–225, 2008. 112

[Ramanujan 2014] M. S. Ramanujan and Saket Saurabh. *Linear Time Parameterized Algorithms via Skew-Symmetric Multicuts.* In SODA, pages 1739–1748, 2014. 21

[Reed 2004] Bruce A. Reed, Kaleigh Smith and Adrian Vetta. *Finding odd cycle transversals.* Oper. Res. Lett., vol. 32, no. 4, pages 299–301, 2004. 21, 22, 42

[Robertson 1984] Neil Robertson and P.D Seymour. *Graph minors III Planar tree-width.* Journal of Combinatorial Theory, Series B, vol. 36, no. 1, pages 49 – 64, 1984. 17

[Robertson 1994] Neil Robertson, Paul D. Seymour and Robin Thomas. *Quickly Excluding a Planar Graph.* J. Comb. Theory, Ser. B, vol. 62, no. 2, pages 323–348, 1994. 187

[Shi 2000] Weiping Shi and Chen Su. *The Rectilinear Steiner Arborescence Problem is NP-Complete*, 2000. 181

[Shigeta 2015] Manami Shigeta and Kazuyuki Amano. *Ordered biclique partitions and communication complexity problems.* Discrete Applied Mathematics, vol. 184, pages 248–252, 2015. 61

[Smorodinsky 2007] Shakhar Smorodinsky. *On the chromatic number of geometric hypergraphs.* SIAM Journal on Discrete Mathematics, vol. 21, no. 3, pages 676–687, 2007. 159

[Spillner 2005] Andreas Spillner. *A Faster Algorithm for the Minimum Weight Triangulation Problem with Few Inner Points.* In Algorithms and Complexity in Durham 2005 - Proceedings of the First ACiD Workshop, 8-10 July 2005, Durham, UK, pages 135–146, 2005. 24

[Tarjan 1985] Robert E. Tarjan. *Decomposition by clique separators.* Discrete Mathematics, vol. 55, no. 2, pages 221 – 232, 1985. 20

[Thomassen 1988] Carsten Thomassen. *On the presence of disjoint subgraphs of a specified type.* Journal of Graph Theory, vol. 12, no. 1, pages 101–111, 1988. 92

[Thomborson 1987] C. Thomborson, L. Deneen and G. Shute. *Computing a rectilinear steiner minimal tree in $n^{O(\sqrt{n})}$ time.* Parallel Algorithms and Architectures, pages 176–183, 1987. 180

[Vapnik 1971] Vladimir N Vapnik and A Ya Chervonenkis. *On the uniform convergence of relative frequencies of events to their probabilities.* Theory of Probability & Its Applications, vol. 16, no. 2, pages 264–280, 1971. 145

[Vikas 2004] Narayan Vikas. Computing and combinatorics: 10th annual international conference, cocoon 2004, jeju island, korea, august 17-20, 2004. proceedings, chapitre Computational Complexity Classification of Partition under Compaction and Retraction, pages 380–391. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. 20

[Wagner 1984] Klaus W. Wagner. *Monotonic Coverings of Finite Sets.* Elektronische Informationsverarbeitung und Kybernetik, vol. 20, no. 12, pages 633–639, 1984. 21, 26, 51

[Yannakakis 1991] Mihalis Yannakakis. *Expressing combinatorial optimization problems by Linear Programs.* Journal of Computer and System Sciences, vol. 43, no. 3, pages 441 – 466, 1991. 61, 67

[Yao 1979] Andrew Chi-Chih Yao. *Some Complexity Questions Related to Distributive Computing(Preliminary Report).* In Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing, STOC '79, pages 209–213, New York, NY, USA, 1979. ACM. 61