Matrix Editing via Multivariate Lens

By

Syed Mohammad Meesum

MATH10201105003

The Institute of Mathematical Sciences, HBNI, Chennai

A thesis submitted to the

Board of Studies in Mathematical Sciences

In partial fulfillment of requirements

For the Degree of

DOCTOR OF PHILOSOPHY

of

HOMI BHABHA NATIONAL INSTITUTE

October, 2017

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 Syed Mohammad Meesum entitled Matrix Editing via Multivariate Lens and recommend that it maybe accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy.

_____ Date:

Chairman -

_____ Date:

Guide/Convener -

_____ Date:

Member 1 -

_____ Date:

Member 2 -

_____ Date:

Member 3 -

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 fulfilment 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 manuscript in 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.

_____

Syed Mohammad Meesum

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.

_____

Syed Mohammad Meesum

List of Publications arising from the thesis

Journal

1. “Parameterized Complexity of Strip Packing and Minimum Volume Packing", Pradeesha Ashok, Sudeshna Kolay, S. M. Meesum, and Saket Saurabh, *Theoretical Computer Science*, **2016**, *Online*.

2. “Reducing Rank of the Adjacency Matrix by Graph Modification", S. M. Meesum, Pranabendu Misra, and Saket Saurabh, *Theoretical Computer Science*, **2016**, *654*, 70–79.

3. “Rank Reduction of Oriented Graphs by Vertex and Edge Deletions", S. M. Meesum and Saket Saurabh, *Algorithmica*, **2017**, 1–20.

Conferences

1. “Matrix Rigidity from the Viewpoint of Parameterized Complexity", Fedor V. Fomin, Daniel Lokshtanov, S. M. Meesum, Saket Saurabh and Meirav Zehavi, 34^{th} *Symposium on Theoretical Aspects of Computer Science*, **2017**, *66*, 32:1–32:14.

Others

1. “Rank Vertex Cover as a Natural Problem for Algebraic Compression", S. M. Meesum, Fahad Panolan, Saket Saurabh and Meirav Zehavi, *Submitted*.

_____

Syed Mohammad Meesum

To my parents.

ACKNOWLEDGEMENTS

First of all I would like to thank my advisor, Dr. Saket Saurabh, without his guidance and far-sightedness this thesis would have remained a dream. The invaluable feedback from my doctoral committee members resulted in simplifying a lot of content present in this thesis. For this I want to thank Dr. Venkatesh Raman and Dr. Vikram Sharma. Most importantly I want to thank them for their patience for listening to my half-baked ideas during doctoral committee meetings and asking questions which helped me understand a lot of things I was working on.

I want to thank my co-authors in different papers Pranabendu Misra, Sudeshna Kolay, Pradeesha Ashok, Daniel Lokshtanov, Meirav Zehavi and Fedor V. Fomin for the interesting discussions. I want to thank Tom, Sudeshna, Roohani, Pranabendu, Mithilesh, Jayakrishnan, Gaurav, Fahad, Daniel, Anantha, Aditi, . . . for many interesting discussions.

I am indebted to my many colleagues who supported me in various forms.

I would like to thank the library staff at IMSc for their support in searching for books and for making the process of procuring books easy. I would also like to thank the administrative and maintenance staff at IMSc for being very helpful.

Lastly, I would like to thank my parents, my siblings and my wife.

Contents

Synopsis

Changing a graph by either deleting vertices/edges or editing edges such that the resulting graph satisfies certain properties or becomes a member of some well-understood graph class is one of the basic problems in graph theory and graph algorithms. These problems are called graph modification problems. A typical graph modification problem takes as an input a graph G, a positive integer k and the objective is to add/delete k vertices (edges) so that the resulting graph belongs to a particular family, $\mathcal{F}$, of graphs.

One of the classical way to define $\mathcal{F}$ is by defining what is called as a *graph property*. A *graph property* Π is a set of graphs, which is closed under isomorphism. The property Π is called *non-trivial* if it includes infinitely many graphs and also excludes infinitely many graphs. The property Π is called *hereditary* if for any graph $G \in \Pi$, all induced subgraphs of G are also present in Π. A *hereditary* property Π always has an *induced forbidden set* characterization i.e. there is a family $\mathsf{Forb}(\Pi)$ of graphs such that, a graph G is in Π if and only if no induced subgraph of G is in $\mathsf{Forb}(\Pi)$. We note that $\mathcal{F}$ can be defined as well by excluding some forbidden *minors* instead of forbidding induced subgraphs.

For any Π which is *non-trivial* and *hereditary*, Lewis and Yannakakis have shown that the corresponding vertex deletion problems are NP-Complete [70, 101]. In addition, for several graph properties the corresponding edge editing problem are known to be NP-Complete as well [14]. This motivates the study of these problems in algorithmic paradigms that are meant for coping with NP-hardness, such as approximation algorithms

and parameterized complexity. We study yet another kinds of graph modification problems in the framework of *parameterized complexity*. It is known that whenever Π has a *finite* induced forbidden set characterization (that is, $|\mathsf{Forb}(\Pi)|$ is finite), the corresponding deletion problem is FPT [16]. Similarly, whenever Π is characterized by a *finite* forbidden set of minors, the corresponding problem is FPT by a celebrated result of Robertson and Seymour [94]. These problems include classical problems such as Vertex Cover, Feedback Vertex Set, Split Vertex Deletion.

Graph modification problems have been at forefront of research in parameterized complexity and several interesting and important results have been obtained recently. In fact, just over the course of the last couple of years there have been results on parameterized algorithms for Chordal Editing [19], Unit Interval Editing [17], Interval Vertex (Edge) Deletion [20, 18], Proper Interval Completion [10], Interval Completion [11], Chordal Completion [42], Cluster Editing [39], Threshold Editing [35], Chain Editing [35], Trivially Perfect Editing [36, 37] and Split Editing [46]. Even more recently, a theory for lower bounds for these problems has also been proposed [9]. We would like to mention that the above list is not comprehensive but rather illustrative.

Changing Rank of a Graph by Modification

In this thesis, we study a class of graph modification problems that are defined algebraically rather than structurally. Given a graph G, two most important matrices that can be associated with it are its adjacency matrix A_G or the corresponding Laplacian L_G. One could study graph modification problems where after editing edges/vertices, the associated matrices of the resulting graph satisfy some algebraic property. This thesis concentrates on these kind of problems. One of the problems we study is when we want the rank of the adjacency matrix of the resulting graph to be at most some fixed constant r. To define these problems formally, for a positive integer r, define Π_r to be the set of graphs G such that the rank of the adjacency matrix A_G is at most r. The rank of the adjacency ma-

trix A_G is an important quantity in graph theory, having connections to many fundamental graph parameters such as the clique number, diameter, domination number etc. [2]. These things motivate the study of following problems.

r-Rank Vertex Deletion	**Parameter:** k
Input: A graph G and a positive integer k	
Question: Can we delete at most k vertices from G so that $rank(A_G) \leq r$?	

For two sets X and Y, we define the symmetric difference as $X \triangle Y = (X \setminus Y) \cup (Y \setminus X)$.

r-Rank Editing	**Parameter:** k
Input: A graph G and a positive integer k	
Question: Can we find a set $F \subseteq V(G) \times V(G)$ of size at most k such that $\mathsf{rank}(A_{G'}) \leq r$, where $G' = (V(G), E(G) \triangle F)$?	

The set F denotes a set of edits to be performed on the graph G. The $\triangle$ operation acts on the sets as follows: if $(u, v) \in F$ and $(u, v) \in E(G)$ then we delete the corresponding edge from G and if $(u, v) \in F$ and $(u, v) \notin E(G)$ then we add the corresponding edge to G. We also consider a variant r-Rank Edge Deletion, where only deletion of edges is allowed.

These problems are also related to some well known problems in graph algorithms. Observe that if $\mathsf{rank}(A_G) = 0$, then G is an empty graph and if $\mathsf{rank}(A_G) = 2$ then G is a complete bipartite graph with some isolated vertices. For $r = 0$, r-Rank Vertex Deletion is the well known Vertex Cover problem. Similarly for $r = 2$, a solution to r-Rank Edge Deletion is a complement of a solution to Maximum Edge Biclique, where the goal is to find a bi-clique subgraph of the given graph with maximum number of edges [87].

We obtained following results about these problems based on structural observations on graphs with low rank adjacency matrix and applications of some elementary methods in parameterized complexity.

1. We first show that all the three problems are NP-Complete.

2. Then we show that these problems are FPT by designing an algorithm with running time $2^{O(k \log r)} n^{O(1)}$ for r-Rank Vertex Deletion, and an algorithm for r-Rank Edge Deletion and r-Rank Editing running in supexponential FPT time $2^{O(f(r)\sqrt{k}\log k)} n^{O(1)}$.

3. Finally, we design polynomial kernels for these problems.

We also extend our results to oriented graphs from undirected graphs in the realm of classical and parameterized complexity. A directed graph $D = (V, E)$, is referred to as an oriented graph, if for any two vertices $u, v \in V(D)$ only one of the arcs between them, either (u, v) or (v, u), is contained in $E(D)$. For an arc $(u, v) \in E(D)$, the $(u, v)^{th}$ entry of the adjacency matrix A_D is defined as $A_D(u, v) = -A_D(v, u) = 1$. If $(u, v) \notin E(D)$, then the corresponding entry is set to zero. This gives us a skew symmetric matrix.

We consider r-Rank Vertex Deletion and r-Rank Edge Deletion when the input is an oriented graph. Similar to the case of undirected graphs, the oriented graph versions of these problem are also related to Vertex Cover, Maximum Edge Biclique, and the concept of "rigidity of matrices" [78]. We obtained similar results for the oriented graph versions of r-Rank Vertex Deletion and r-Rank Edge Deletion as mentioned earlier for the case of undirected graphs. To obtain the results, we show that the "reduced" oriented graphs of rank r have size at most 3^r. This result could be of independent interest. This together with standard methods in parameterized complexity easily imply FPT and kernel result for r-Rank Vertex Deletion. To obtain sub-exponential algorithm for r-Rank Edge Deletion we do a modification to an algorithm of Damaschke et. al. [29] and use it as a subroutine.

Matrix Rigidity

The concept which generalizes the problems discussed above is that of *matrix rigidity*. The problems we studied earlier [80, 81] are related to Matrix Rigidity but are different from it in the way that the input matrices are restricted to being either symmetric or skew-symmetric. Another difference is that whenever we modify an entry (u, v) in the matrix

we are forced to modify the corresponding $(v, u)^{th}$ entry as well. In the matrix rigidity problem one gets rid of both of these restrictions along with the restriction on the field over which the input matrix is provided. We next provide the necessary definitions and motivation for this problem.

The *rigidity of a matrix* is a classical concept in Computational Complexity Theory, which was introduced by Grigoriev [49, 50] in 1976 and by Valiant [99] in 1977. Constructions of rigid matrices are known to imply lower bounds of significant importance relating to arithmetic circuits. Yet, from the viewpoint of Parameterized Complexity, the study of central properties of matrices in general, and of the rigidity of a matrix in particular, has been neglected. We conduct a comprehensive study of different aspects of the computation of the rigidity of *general matrices* in the framework of Parameterized Complexity.

Formally, given a matrix A over a field $\mathbb{F}$, the rigidity of A, denoted by $\mathcal{R}_A^{\mathbb{F}}(r)$, is defined as the minimum Hamming distance between A and a matrix of rank at most r. In other words, $\mathcal{R}_A^{\mathbb{F}}(r)$ is the minimum number of entries in A that should to be edited in order to obtain a matrix of rank at most r. Naturally, given a parameter k, the Matrix Rigidity problem asks whether $\mathcal{R}_A^{\mathbb{F}}(r) \leq k$. The case when $\mathbb{F} = \mathbb{Q}$ or the edited entries must contain integers, it is not known whether the problem is decidable [95]. Thus, we focus on the cases where $\mathbb{F} = \mathbb{R}$ or $\mathbb{F}$ is any finite field and study the following forms of Matrix Rigidity.

Real Matrix Rigidity	**Parameter:** r, k
Input: A matrix A with integer entries, and integers $r, k \in \mathbb{Z}_+$.	
Question: Is $\mathcal{R}_A^{\mathbb{R}}(r) \leq k$?	

The finite field case of FF Matrix Rigidity problem includes $\mathbb{F}_p$ as part of the input.

FF Matrix Rigidity	**Parameter:** p, r, k
Input: An order p finite field $\mathbb{F}_p$, a matrix A over $\mathbb{F}_p$, and integers $r, k \in \mathbb{Z}_+$.	
Question: Is $\mathcal{R}_A^{\mathbb{F}_p}(r) \leq k$?	

Valiant [99] presented the notion of the rigidity of a matrix as a means to prove lower

bounds for linear algebraic circuits. He showed that the existence of an $n \times n$ matrix A with $\mathcal{R}_A^{\mathbb{F}}(\epsilon n) \geq n^{1+\delta}$ would imply that the linear transformation defined by A cannot be computed by any arithmetic circuit having size $O(n)$ and depth $O(\log n)$ in which each gate of the circuit computes a linear combination of its inputs. Later, Razborov [90] (see [72]) established relations between lower bounds on rigidity of matrices over the reals or finite fields and strong separation results in Communication Complexity. Although many efforts have been made in this direction [43, 97, 73, 67] (not exhaustive), proofs of quadratic separation lower bounds for explicit families of matrices still remains elusive. For a recent survey see [74]. The formulation of the Matrix Rigidity as stated in this thesis was first considered by Mahajan and Sarma [78], and it was shown to NP-hard for any field by Deshpande [31]. We established that both Real Matrix Rigidity and FF Matrix Rigidity are FPT with respect to $r + k$. Specifically, we proved the following.

Theorem 1. Real Matrix Rigidity *can be solved in time* $O^*(2^{O((r+k)\cdot\log(r\cdot k))})$.

Theorem 2. FF Matrix Rigidity *can be solved in time* $O^*(f(r,k))$ *for a function* f *that depends only on* r *and* k.

The dependency of the running times on the dimension of the input matrix is polynomial, and in the case of FF Matrix Rigidity, the dependency of the running time on p is also polynomial. Interestingly, in the case of Real Matrix Rigidity, the dependency of the running time on the maximum bit-length of any entry in *both* input and output matrices is polynomial! We find these results quite surprising, particularly due to the fact that in case $\mathbb{F} = \mathbb{Q}$ or the edited entries must contain integers, it is not even known whether Matrix Rigidity is decidable [95]. We also show that,

Theorem 1. *The* FF Matrix Rigidity *problem is solvable in time* $O^*(2^{O(f(r,p)\sqrt{k}\log k)})$ *for some function* f *that depends only on* r *and* p.

Here, the dependency of the running time on k is subexponential, but the dependency of the running time on p is unsatisfactory, in case p is not fixed. This algorithm is based on a

technically involved adaptation of ideas already used in the papers [80, 81]. To obtain our main results, we develop a simple to describe dimension reduction procedure. Given an instance of Matrix Rigidity, it outputs (in polynomial time) an equivalent instance where the matrix contains at most $O((r \cdot k)^2)$ entries. Furthermore, the entries of the output matrix are a subset of the entries of the input matrix. This procedure establishes that FF Matrix Rigidity admits a polynomial kernel with respect to $r + k + p$. Our procedure is modular—it handles rows and columns in separate phases.

Using the dimension reduction procedure, we tackle Real Matrix Rigidity and FF Matrix Rigidity by employing central tools in Algebraic Geometry. For this purpose, we first recall that the rank of a matrix is at most r if and only if the determinant of all of its $(r + 1) \times (r + 1)$ submatrices is 0. Since at this point we can assume that we have a matrix containing only $O((r \cdot k)^2)$ entries at hand, we may "guess" *which* entries should be edited. Yet, it is not clear *how* these entries should be edited. However, with the above observation in mind, we are able to proceed by viewing our current problem in terms of an algebraic variety. In particular, this viewpoint gives rise to the applicability of firmly established tools that determine the feasibility of a system of polynomials [7, 62].

Our main results are complemented by a W[1]-hardness result. Our earlier papers [80, 81] already imply that both of these problems are para-NP-hard with respect to the parameter r. To complete the picture, we show that both Real Matrix Rigidity and FF Matrix Rigidity are W[1]-hard with respect to the parameter k using a series of four reductions. We first define a special case of Odd Set, which we call Special Odd Set, and observe that its W[1]-hardness follows from the proof of the W[1]-hardness of Odd Set that is given in [33]. The correctness of our reductions crucially relies on the implications of the properties of this special case. Our first reduction translates Special Odd Set to a problem involving matrices rather than sets, which we call Special Odd Matrix. Then, to be able to discuss any finite field as well as the field of reals, we introduce new variants of Special Odd Matrix and the Nearest Codeword problem, called $\mathbb{F}$-Odd Matrix and

$\mathbb{F}$-Nearest Codeword, respectively. The application of our second reduction results in an instance of $\mathbb{F}$-Odd Matrix. Then, the application of our third reduction, which builds upon [12], results in an instance of $\mathbb{F}$-Nearest Codeword. Finally, we devise a reduction whose application results in an instance of Matrix Rigidity.

Above Guarantee Algebraic Compression of Vertex Cover

We also study the problem of compression of Vertex Cover instances above the guarantee of maximum matching. Formally, given a graph H and a parameter k, the Vertex Cover Above MM problem asks whether H admits a vertex cover of size at most $\mu(H)+k$, where $\mu(H)$ is the maximum size of a matching of H, and the Vertex Cover Above LP problem asks whether H admits a vertex cover of size at most $\ell(H)+k$, where $\ell(H)$ is the fractional vertex cover number of H.

Several parameterized algorithms for these two problems have been developed in the last decade [91, 89, 28, 84, 75]. Currently, the best known algorithm for Vertex Cover Above LP, which is also the best known algorithm Vertex Cover Above MM, runs in time $2.3146^k \cdot n^{O(1)}$ [75]. On the other hand, the question of the existence of polynomial kernelizations of these two problems has been a longstanding, notorious open problem in Parameterized Complexity. Five years ago, the breakthrough work by Kratsch and Wahlström on representative sets has finally answered this question in the affirmative [66]. Up to date, the kernelizations by Kratsch and Wahlström have remained the only known (randomized) polynomial compressions of Vertex Cover Above MM and Vertex Cover Above LP. Note that since $\ell(H)$ is necessarily at least as large as $\mu(H)$, a polynomial compression of Vertex Cover Above LP also doubles as a polynomial compression of Vertex Cover Above MM. We also remark that several central problems in Parameterized Complexity, such as the Odd Cycle Transversal problem, are known to admit parameter-preserving reductions to Vertex Cover Above LP [75]. Hence, the significance of a polynomial compression of Vertex Cover Above LP also stems from the observa-

tion that it simultaneously serves as a polynomial compression of additional well-known problems, and can therefore potentially establish the target problem as a natural candidate to express compressed problem instances.

Recently, a higher above-guarantee parameterization of Vertex Cover, resulting in the Vertex Cover Above Lovász-Plummer, has been introduced by Garg and Philip [45]. Here, given a graph H and a parameter k, the objective is to determine whether H admits a vertex cover of size at most $(2\ell(H) - \mu(H)) + k$. Garg and Philip [45] showed that this problem is solvable in time $3^k \cdot n^{O(1)}$, and Kratsch [65] showed that it admits a (randomized) kernelization that results in a large, yet polynomial, kernel. We remark that above-guarantee parameterizations can very easily reach bars beyond which the problem at hand is no longer FPT. For example, Gutin et al. [52] showed that the parameterization of Vertex Cover above $m/\Delta(H)$, where $\Delta(H)$ is the maximum degree of a vertex in H and m is the number of edges in H, results in a problem that is not FPT (unless FPT =W[1]).

We present an alternative, *algebraic compression* of the Vertex Cover Above LP problem into the Rank Vertex Cover problem. We remark that Rank Vertex Cover was originally introduced by Lovász as a tool for the examination of critical graphs [76]. Given a graph H, a parameter ℓ, and a bijection between $V(G)$ and the set of columns of a representation of a matroid M, the objective of Rank Vertex Cover is to find a vertex cover of H whose rank, which is defined by the set of columns corresponding to its vertices, is upper bounded by ℓ.

We obtain a (randomized) polynomial compression of size $\tilde{O}(k^7 + k^{4.5}\log(1/\varepsilon))$[1], where ε is the probability of failure. Here, by failure we mean that we output an instance of Rank Vertex Cover whose size is not within the claimed bound or that is not equivalent to the input instance. In the first case, we can simply discard the output instance, and return an arbitrary instance of constant size; thus, we ensure that failure only refers to the maintenance of equivalence. Our work makes use of properties of linear spaces and

[1] $\tilde{O}$ hide factors polynomial in $\log k$

matroids, and also relies on elementary probability theory.

Parameterized Complexity of Strip Packing

The problem of packing objects optimally inside a bin/box is studied in various forms. Two prominent examples are the Bin Packing problem and the Knapsack problem. We study generalizations of these problems in a geometric setting. A natural generalization of the Knapsack problem to two dimensions is the Cutting Stock problem [8]. More specifically, we study the problem of packing axis-parallel rectangles inside a rectangular box when only translations are allowed. This restriction of not allowing rotations does not make the problem artificial as it still finds practical application [71]. These problems also find applications in VLSI design [83], scheduling [100], packing television commercial into station breaks [13] etc.

We consider two versions of this problem. In the Strip Packing problem, given a set of n axis-parallel rectangles and a rectangular box (strip) of width W, the goal is to pack all rectangles into this strip so that the height used is minimized. In the Minimum Volume Packing problem, given a set of n axis-parallel rectangles, goal is to pack these rectangles in a rectangular container so that the volume of the container is minimized. We also study these problems in higher dimensions. Formally, the problems are defined as follows:

d-Strip Packing **Input:** A list of boxes $\{b_i \in \mathbb{N}^d \mid 1 \leq i \leq n\}$ and a vector of positive integers $W \in \mathbb{N}^{d-1}$. **Output:** The minimum integer k such that all the boxes can be packed under translation into a strip with dimension vector $W \times k$, k being the height of the strip.

Volume packing in two-dimensions is an active area of research due to its immense practical importance [56].

d-Volume Packing

Input: A list of n boxes $\{b_i \in \mathbb{N}^d \mid 1 \leq i \leq n\}$.

Output: The minimum volume rectangular container into which the input boxes can be packed under translation.

The decision versions of both these problems are proved to be NP-Complete and are well studied in the context of approximation algorithms. Two-dimensional Strip Packing admits an AFPTAS [63]. Since Bin Packing, which is a special case of Strip Packing, cannot have a PTAS [4], the same holds for Strip Packing. For recent approximation algorithms and results on Minimum Volume Packing see [3]. The online version of these problems are also studied [53, 102, 59]. For a survey of these problems we refer the reader to [71, 26].

We study Strip Packing and Minimum Volume Packing from the perspective of *parameterized complexity*, and consider the standard parameterized version of these problems, namely the case when the parameter is the size of the solution. To the best of our knowledge, this is the first study on the parameterized version of these problems. For the Strip Packing problem, we prove that the parameterized version is W-hard for the general case. However, the problem becomes FPT for a special case where the dimensions of the boxes and therefore the number of types of boxes is bounded by a constant, this special case is also inspired by a variant of the Bin Packing problem with similar constraints [47]. We also consider several special kind of input box dimensions where PTAS and even polynomial time algorithms exist.

We obtained the following results:

1. We prove that 2-Minimum Volume Packing is in FPT by giving an algorithm with running time $(2 \cdot \sqrt{2})^k \cdot k^{O(1)}$. This algorithm can be generalized to d-dimensions.

2. We prove that Strip Packing is W-hard even in two dimensions.

3. We consider a special case of Strip Packing where every dimension of the input boxes is bounded by a constant. For this case, we show that Strip Packing problem admits an FPT algorithm.

4. We consider some special cases of Strip Packing where the input rectangles are squares whose dimensions come from a special set and show that they are solvable in polynomial time.

List of Figures

Chapter 1

Introduction

This chapter provides an outline of the thesis and serves as a collection of various definitions which are the underlying principles used everywhere in this thesis. The framework for all our analysis is the concept of parameterized complexity. It is different than the classical complexity in the sense that it analyses the problems with respect to various parameters associated with a problem. For instance, for an NP-Complete decision problem, a parameterized version can effectively point out the source of hardness in the problem. This informal notion is used a lot in parameterized complexity and shows us that various NP-Complete problems are not equivalent in terms of hardness. There are several approaches to deal with NP-Hard problems. When we want an approximate solution with a guarantee on the quality of solution we turn to approximation algorithms, the main advantage of these algorithms is that we get an approximate solution in time which is a polynomial in input size. Often it is important in practical applications to solve the problem exactly, for this we turn to the paradigm of parameterized complexity.

In practice, for a computationally hard problem a lot of input instances can be solved very quickly. This could be due to various reasons, one being the presence of certain structure in the input instance which can be exploited to solve the problem faster than brute force. Intuitively, an extra parameter is provided as input which quantifies some

structural complexity of the input instance of a problem. A very well known example is the tree-width of a graph. The Vertex Cover problem can be efficiently solved on a tree, one of the natural 'distance' function of any graph from a tree is the tree-width of the graph and it is possible to get a parameterized algorithm for the Vertex Cover with tree-width as the parameter. As the tree-width of graphs increases (or as the graph becomes far from being tree like) the running time of finding a vertex cover increases. A lot of research has focused on considering a subset of input instances or special input classes of a hard problem and giving efficient algorithms for them. In parameterized complexity, a similar approach has proved very useful, which consists of defining 'distance' to a an efficiently solvable input class and treating that as a parameter. Another parameter used a lot is the size of the output. In this thesis, one of the parameters we consider is the rank of the adjacency matrix as a measure of the structural complexity of a graph.

1.1 Outline of the Thesis

The three chapters 2, 3 and 4 study the modification problem of reducing the rank of matrices to a target rank r when at most k operations are allowed on the matrices. These chapters are the main theme of this thesis. We start with undirected graphs in chapter 2. We start with the adjacency matrix of an undirected graph where modification is done in a special way on the matrices as these operations correspond to graph operations like vertex deletions or edge editing. The adjacency matrix of a graph is symmetric and its entries are restricted to take only zero/one values. Moreover, the vertex deletion operation corresponds to deletion of a column and the corresponding row from the adjacency matrix, the modified matrix always stays symmetric. In the case of edge editing, whenever we change the (i, j) entry of the adjacency matrix we also set the (j, i) entry of the matrix to the same value. Similarly, chapter 3 studies these problems on oriented graphs. Finally, in chapter 4 we work with general matrices, the entries in the matrix can be arbitrary and the matrix is no longer restricted to being symmetric or skew-symmetric.

The second last chapter, chapter 5, deals with compressing instances of vertex cover in which the compressed instance size is guaranteed to be at most a polynomial in the above guarantee parameter above the maximum matching size. This is achieved via an intermediate problem which defines a notion of rank of a vertex cover.

Finally, chapter 6 deals with the strip packing problem, which has received a lot of attention recently mostly in the approximation algorithms paradigm. We study its behavior with respect to various parameters.

1.2 Notations and Definitions

The set of natural numbers $\{1, \ldots, n\}$ will be denoted using $[n]$. The set of integers and reals are denoted using $\mathbb{Z}$ and $\mathbb{R}$ respectively. The set of non-negative integers is denoted using $\mathbb{Z}_+$. We shall use the letters $G, H, \ldots$ to denote graphs. A family of sets over a universe set U consists of subsets of U. A family of sets is said to be k-uniform if each element in it has cardinality equal to k. A k-uniform family over the universe U shall be denoted using the symbol $\binom{U}{k}$.

We shall refer to an algorithm as being efficiently computable if the number of steps taken by it is a polynomial in the length of the input instance.

1.3 Graphs

A *directed graph* is a structure consisting of the tuple (V, E), where V is a finite set of vertices and $E \subseteq V \times V$ is the set of edges. A vertex $u \in V$ is said to have a 'loop' if (u, u) is an edge. We would be concerned with loopless directed graphs.

An *undirected graph* is a graph in which for any edge $(u, v) \in E$, the 'reverse' edge (v, u) is also contained in E. Another way to define an undirected graph is by restricting its edges

to be a subset of $\binom{V}{2}$ i.e. a edges are sets having two vertices. We would be dealing with loopless undirected graphs, which are also known as simple graphs. The graph definition can be relaxed to allow for multiple edges between vertices, such a graph is referred to as a multigraph. We shall use the letters $G, H, \ldots$ to denote undirected graphs.

We now list out some important definitions related to a graph $G = (V, E)$.

Excluded Neighborhood of a Vertex Given a vertex $u \in V$, its excluded neighbourhood $N(u)$ is equal to the set $\{v : (u, v) \in E\}$.

Included Neighborhood of a Vertex For a vertex $u \in V$, it is equal to $N(u) \cup \{u\}$ and is denoted using $N[u]$.

Degree of a Vertex For a vertex $u \in V$, it is denoted using d_u and is equal to $|N(u)|$.

Adjacency Matrix An undirected graph G having n vertices can be represented using an $n \times n$ table in which $(i, j)^{th}$ entry is one if (i, j) is an edge otherwise it is zero. This table can also interpreted as a matrix and is referred to as the adjacency matrix of the graph G.

Subgraph A graph $G' = (V', E')$ is said to be a subgraph of a graph $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$.

Induced Subgraph A subgraph $G' = (V', E')$ is said to be an induced subgraph of $G = (V, E)$ if E' consists of all the edges between vertices in V' which are contained in E. As an induced graph of G over a vertex set depends only on G and V', it is denoted using $G' = G[V']$.

Connected Graph A graph $G = (V, E)$ is said to be connected if between any pair of vertices in V there exists a path connecting them. A path is a sequence of vertices $u_1, \ldots, u_\ell$ such that (u_i, u_{i+1}) is an edge for $i \in [\ell - 1]$.

Clique A set $U \subseteq V$ is said to be a clique in $G = (V, E)$ if every pair of elements of U

is adjacent to each other. Thee clique number of G is the number of vertices in a largest clique in G. A clique on three vertices is also called a triangle.

Independent Set A set $U \subset V$ is said to be an independent set in $G = (V, E)$ if no two elements of U are adjacent to each other. The independence number of G is the number of vertices in a largest independent set in G.

Vertex Cover A set $U \subseteq V$ is said to be a vertex cover of $G = (V, E)$ if every edge in G is adjacent to at least one vertex in U.

Isthmus/Leaf A vertex $u \in V(G)$ is called as an isthmus or a leaf if its degree is exactly one.

1.4 Parameterized Complexity Primer

Given a problem Π the decision version consists of deciding if an instance x is Yes or No. Therefore, each problem Π can be thought of as a set and the the decision question is one of deciding the membership of an instance in this set Π. The problem Π can be *parameterized* by assigning a number k, which is called the *parameter* and represents some property of the instance. For example, a typical parameter is the size of the optimum solution to the instance, as this is the most common way to parameterize a problem we shall refer to it as the *standard parameterization*. We define a parameterized problem as follows.

Definition 1. *A* parameterized *problem is a set* $\Pi \subseteq \Sigma^* \times \mathbb{N}$*, where* Σ *is a finite alphabet. For an instance* $(x, k) \in \Sigma^* \times \mathbb{N}$ *of a parameterized problem, we refer to* x *as the input and* k *is referred to as the parameter.*

The definition above can be extended to problems where there are many more parameters. For example, a problem $\Pi \subseteq \Sigma^* \times \mathbb{N}^p$ is a problem with p many parameters. The problem

Vertex Cover when parameterized by the output size k admits an algorithm with running time $2^k n^{O(1)}$, where n is the length of the input graph.

A parameterized problem is called as *fixed parameter tractable* (FPT), if a parameterized instance (x, k) of the problem is solvable in time $f(k)n^{O(1)}$, where n is the size of the instance. Formally, we define the idea of fixed parameter tractability as follows.

Definition 2. *A parameterized problem* $\Pi \subseteq \Sigma^* \times \mathbb{N}$ *is* fixed parameter tractable *(FPT) if there is a computable function* $f : \mathbb{N} \to \mathbb{N}$*, a constant* $c \in \mathbb{N}$*, and an algorithm which decides the membership of an instance* (x, k) *in* Π *using at most* $f(k) \cdot |x|^c$ *steps.*

1.4.1 Kernelization

Often the input instance x for a problem Π can be pruned or preprocessed to produce an equivalent instance depending on the value of the parameter k. Suppose the preprocessing algorithm runs in time which is a polynomial function of $|x|$ and k. Its output is another instance (x', k') of Π which is *equivalent* to (x, k) in the sense that (x, k) is a Yes-instance of Π if and only if (x', k') is a Yes-instance of Π. In many cases, the preprocessing itself may solve the problem by outputting an instance for which it is very easy to verify its membership in Π. As the preprocessing is allowed to take polynomial number of steps, all the instances of an NP-Hard problem can not be solved by it. Thus, the preprocessing algorithm has to stop at some point when the preprocessing rules are no longer applicable on the input. If the output (x', k') of the preprocessing algorithm always satisfies that $|x'| \leq g(k)$ and $k' \leq g(k)$ for some function g which depends on k then the problem Π is said to admit a *kernel size* $g(k)$ and the output instance x' is referred to as the *problem kernel*. We condense this explanation into one sentence to get the following.

A parameterized problem admits a *polynomial kernel* if there is a polynomial time algorithm that given an instance of the problem, outputs an equivalent instance whose size is bounded by a polynomial in the parameter.

We provide the definition formally as follows.

Definition 3 (Kernelization)**.** *A kernelization algorithm for a parameterized problem* $\Pi \subseteq \Sigma^* \times \mathbb{N}$ *is an algorithm that, given* $(x, k) \in \Sigma^* \times \mathbb{N}$*, outputs, in time polynomial in* $(|x| + k)$*, a pair* $(x', k') \in \Sigma^* \times \mathbb{N}$ *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* $g(k) = k^{O(1)}$*, then we say that* Π *admits a polynomial kernel.*

In many cases the problems are not FPT. For example, the problem Graph Colorability which requires one to find if a given graph G is properly-colorable using k colors. Suppose, it admits an FPT algorithm of the form $f(k) \cdot |G|^{O(1)}$. It is known that the problem 3-Colorability is NP-Complete as well. Therefore, using the claimed FPT algorithm for Graph Colorability it would be possible to solve 3-Colorability in time which is a polynomial in graph size, contradicting the widely held assumption that P $\neq$ NP.

A lot of problems which seemingly do not admit an FPT algorithm have no known way to be reduced to the classical complexity theoretic questions like P $\neq$ NP. A series of complexity classes were introduced above FPT by Downey and Fellows [32] which are helpful for proving a problem to be not in FPT. One such class hierarchy is the W-hierarchy, it is a series of classes satisfying W[1] $\subseteq$ W[2] $\subseteq \cdots$. We would mostly be concerning ourselves with using this hierarchy for the purpose of proving that a problem is not FPT based on the assumption that FPT $\neq$ W[1]. In classical complexity the notion of "hardness" of a problem with respect to the class NP is defined using polynomial time reductions. Analogous to that we have a notion of parameterized reductions.

Definition 4. *Let* $\Pi \subseteq \Sigma^* \times \mathbb{N}$ *and* $\Pi' \subseteq \Sigma^* \times \mathbb{N}$ *be parameterized problems. An FPT-reduction from* Π *to* Π' *is an algorithm that computes for every instance* (x, k) *of* Π *an instance* (x', k') *of* Π' *in time* $f(k) \cdot |x|^{O(1)}$ *such that* $k' \leq g(k)$ *and*

$$(x, k) \in \Pi \iff (x', k') \in \Pi' \tag{1.1}$$

(for computable functions $f, g : \mathbb{N} \to \mathbb{N}$ and a constant $c \in \mathbb{N}$).

To prove that a problem Π does not admit a FPT algorithm we reduce a known W-Hard problem Γ to it using a parameterized reduction. This suffices to show that Π is unexpected to have an FPT algorithm as that would imply that Γ is in FPT.

1.4.2 Bounded Search Trees

Informally, a *bounded search tree* or *branching* is used to represent the execution of an algorithm which solves a problem based on the solution of sub-problems. It can be represented as a tree and the algorithm can be imagined to solve the sub-problems one at a time by traversing this tree. The correctness of a branching algorithm can be justified by arguing that in the case of a Yes-instance some sequence of decisions captured by the algorithm leads to a feasible solution. The running time of the algorithm is given by the size of the branching tree. For a parameterized instance, if the size of the branching tree is bounded by a function of the parameter and each step of the algorithm takes polynomial time then such a branching algorithm leads to an FPT algorithm.

One method to bound the size of a branching tree employs the notion of *branching vectors*. To each node of the tree we associate a value using a function which depends on the instance to be solved at that node. This function, usually referred to as a *measure function*, is set up in such a way that it takes a smaller value for a sub-problem. It should also satisfy the property that it is a bounded function. Now the size of the tree can be upper-bounded by looking at the drop in value of the measure function at each branch of a node. This drop is represented as a tuple of numbers and is referred to as a *branching vector*. Formally, suppose that the algorithm executes a rule which has ℓ branches (each corresponding to a recursive call), such that in the i^{th} branch, the current value of the measure decreases by b_i. Then, $(b_1, b_2, \ldots, b_\ell)$ is the branching vector of this rule. We say that α is the *branching number* of $(b_1, b_2, \ldots, b_\ell)$ if it is the (unique) positive real root of

$x^{b^*} = x^{b^*-b_1} + x^{b^*-b_2} + \ldots + x^{b^*-b_\ell}$, where $b^* = \max\{b_1, b_2, \ldots, b_\ell\}$. If $r > 0$ is the initial value of the measure, and the algorithm returns a result when (or before) it becomes negative, the running time is bounded by $O^*(\alpha^r)$. For more details we refer the reader to [27].

1.5 Linear Algebra

The use of linear algebra in the thesis would mostly be concerned with the rank of matrices and representable matroids. To define a vector space $\mathcal{V}$ we need two ingredients, one is a field $\mathbb{F}$ and a the other is a set of two binary operators which act on the elements of a vector space. The elements of $\mathbb{F}$ are referred to as scalars and the elements of $\mathcal{V}$ are referred to as vectors. The two binary operators acting on $\mathcal{V}$ are referred to as addition and multiplication and are denoted using standard mathematical notations for these operations. For example, the sum of two vectors $u, v \in \mathcal{V}$ is denoted using $u + v$ and the product of a vector v with a scalar $a \in \mathbb{F}$ is denoted using av. The vector space $\mathcal{V}$ forms an abelian group along with the addition operator. Additionally, the scalars and vectors satisfy the following properties:

1. $c(u + v) = cu + cv$,
2. $(a + b)u = au + bu$, where $a, b \in \mathbb{F}$,
3. $a(bu) = (ab)u$,
4. $1v = v$, where 1 is the multiplicative identity in $\mathbb{F}$.

For a vector space $\mathcal{V}$, a subspace of $\mathcal{V}$ consists of a subset of vectors of $\mathcal{V}$ which form a vector space.

Given a set of vectors $V = \{v_1, \ldots, v_n\}$, the set V is said to be linearly dependent if there exists a set of scalars $\{a_1, \ldots, a_n\}$, not all zero, such that $\sum_{i=1}^{n} a_i v_i = 0$; otherwise the set V is said to be linearly independent. The dimension of a set of linearly independent vectors

is equal to its cardinality. Given a set of vectors V, its *rank* is defined to be the dimension of its maximal linearly independent subset. It can be easily seen that the rank of V is a uniquely defined number.

The dimension of a vector space $\mathcal{V}$ is defined to be the maximum value of rank over all the subsets of vectors of $\mathcal{V}$. Let d be the dimension of a vectors space $\mathcal{V}$ over a field $\mathbb{F}$. We would be concerned with vector spaces with vectors drawn from the set $\mathbb{F}^d$. It can be checked that its rank is d.

A matrix A of size $r \times c$ over a field $\mathbb{F}$ is a collection of scalars a_{ij}, for $i \in [r]$ and $j \in [c]$. We can interpret a matrix A as a collection of vectors also. The i^{th} row of A is the vector $A_i = (a_{i1}, \ldots, a_{ic})$. The j^{th} column of A is the vector $A^j = (a_{1j}, \ldots, a_{rj})$. The column rank of A over a vector space $\mathcal{V}$ over $\mathbb{F}$ is defined to be the rank of its columns $\{A^1, \ldots, A^c\}$. The row rank can be defined similarly. It can be proved that the row rank and column rank of a matrix are equal. Thus, we refer to the row rank or column rank of a matrix as the rank of the matrix.

Let A be a square matrix of size $n \times n$. We denote the $(i, j)^{th}$ element of a matrix A by $A_{i,j}$. Let S_n denote the set of all permutations over n elements and let sign denote the sign function of a permutation. The determinant of a matrix is defined as the following quantity:

$$\mathrm{Det}(A) = \sum_{\sigma \in S_n} \mathrm{sign}(\sigma) \prod_{i=1}^{n} A_{i,\sigma(i)}. \tag{1.2}$$

The determinant can be used to define the rank of a matrix. For an $r \times c$ matrix A, let $I \subseteq [r]$ and $J \subseteq [c]$ be subsets of row and column indices of A respectively. A submatrix $A_{I,J}$ of A is a matrix which consists of the elements $(A_{i,j})$, where $i \in I$ and $j \in J$. The rank of a matrix A can also be defined in terms of the determinant of square submatrices of a given matrix.

Lemma 1. *The rank of a matrix A is equal to the maximum of the number of rows of a*

square submatrix of A having non-zero determinant.

We shall find use of the above definition of rank in later chapters even though computing the rank of a matrix using above lemma is computationally inefficient. As can be easily seen the rank of a matrix can be computed in polynomial number of steps in input size by finding the maximum size linearly independent set of rows of a matrix. To find the rank using the above lemma one needs to enumerate all the square submatrices and evaluating the determinant which needs checking an exponential number of submatrices in the input matrix.

Chapter 2

Reducing Rank of the Adjacency Matrix by Graph Modification[1]

The main topic of this chapter is to study a class of graph modification problems. A typical graph modification problem takes as input a graph G, a positive integer k and the objective is to add/delete k vertices (edges) so that the resulting graph belongs to a particular family, $\mathcal{F}$, of graphs. In general the family $\mathcal{F}$ is defined by forbidden subgraph/minor characterisation. In this chapter rather than taking a structural route to define $\mathcal{F}$, we take algebraic route. More formally, given a fixed positive integer r, we define $\mathcal{F}_r$ as the family of graphs where for each $G \in \mathcal{F}_r$, the rank of the adjacency matrix of G is at most r. Using the family $\mathcal{F}_r$ we initiate algorithmic study, both in classical and parameterized complexity, of following graph modification problems: r-Rank Vertex Deletion, r-Rank Edge Deletion and r-Rank Editing. These problems generalize the classical Vertex Cover problem and a variant of the d-Cluster Editing problem. We first show that all the three problems are NP-Complete. Then we show that these problems are fixed parameter tractable (FPT) by designing an algorithm with running time $2^{O(k \log r)} n^{O(1)}$ for r-Rank Vertex Deletion, and an algorithm for r-Rank Edge Deletion and r-Rank Editing run-

[1]This chapter is based on a joint work with Pranabendu Misra and Saket Saurabh [80].

ning in time $2^{O(f(r)\sqrt{k}\log k)}n^{O(1)}$. We complement our FPT result by designing polynomial kernels for these problems.

2.1 Introduction

A typical graph modification problem takes as an input a graph G, a positive integer k and the objective is to add/delete k vertices (edges) so that the resulting graph belongs to a particular family, $\mathcal{F}$, of graphs. One of the classical way to define $\mathcal{F}$ is by defining what is called as a *graph property*. A *graph property* Π is a set of graphs, which is closed under isomorphism. The property Π is called non-trivial if it includes infinitely many graphs and also excludes infinitely many graphs. The property Π is called *hereditary* if for any graph $G \in \Pi$, all induced subgraphs of G are also present in Π. A hereditary property Π always has an *induced forbidden set* characterization i.e. there is a family Forb(Π) of graphs such that, a graph G is in Π if and only if no induced subgraph of G is in Forb(Π). Another way of defining $\mathcal{F}$ is by excluding some forbidden *minors*. A graph H is said to be a *minor* of a graph G if H can be obtained from G by deletion of some edges and vertices or contraction of some edges. A hereditary property Π has a *forbidden minor* characterization if there is a family Forb(Π) of graphs such that no graph in Π has a graph in Forb(Π) as a minor. For a graph property Π, the Π-Graph Modification problem is defined as follows: given a graph G and a positive integer k, can we delete (edit) at most k vertices (edges) so that the resulting graph G' is in Π?

For any Π which is non-trivial and hereditary, Lewis and Yannakakis have shown that the corresponding vertex deletion problems are NP-Complete [70, 101]. In addition, for several graph properties the corresponding edge editing problem are known to be NP-Complete as well [14]. This motivates the study of these problems in algorithmic paradigms that are meant for coping with NP-hardness, such as approximation algorithms and parameterized complexity. In this chapter, we study yet another kinds of graph mod-

ification problems in the framework of *parameterized complexity*. It is known that whenever Π has a *finite* induced forbidden set characterization (that is, $|\mathsf{Forb}(\Pi)|$ is finite), the corresponding deletion problem is FPT [16]. Similarly, whenever Π is characterized by a *finite* forbidden set of minors, the corresponding problem is FPT by a celebrated result of Robertson and Seymour [94]. These problems include classical problems such as Vertex Cover, Feedback Vertex Set, Split Vertex Deletion. There has also been extensive study of graph modification problems when the corresponding $\mathsf{Forb}(\Pi)$ is not finite, such as Chordal Vertex Deletion, Interval Vertex Deletion, Chordal Completion and Odd Cycle Transversal [20, 42, 92].

In this chapter, we step aside and study a class of graph modification problems that are defined algebraically rather than structurally. Given a graph G, two most important matrices that can be associated with it are its adjacency matrix A_G and the corresponding Laplacian L_G. One could study graph modification problems where after editing edges/vertices, the resulting graph has a certain kind of eigenvalue spectrum or has a certain upper bound on the second eigenvalue or the corresponding adjacency matrix satisfies certain properties. The topic of this chapter concerns modifying the adjacency matrix of an undirected graph. In particular, we want the rank of the adjacency matrix of the resulting graph to be at most some fixed constant r. To define these problems formally, for a positive integer r, define Π_r to be the set of graphs G such that the rank of the adjacency matrix A_G is at most r. The rank of the adjacency matrix A_G is an important quantity in graph theory. It also has connections to many fundamental graph parameters such as the clique number, diameter, domination number etc. [2]. All these motivate the study of the following problems.

r-Rank Vertex Deletion	**Parameter:** k
Input: A graph G and a positive integer k	
Question: Can we delete at most k vertices from G so that $rank(A_G) \leq r$?	

We use $\triangle$ to denote the standard symmetric difference of sets. For two sets X and Y, we define $X \triangle Y = (X \setminus Y) \cup (Y \setminus X)$, i.e. the set of all elements which are exactly in X or Y

but not both.

r-Rank Editing	**Parameter:** k
Input: A graph G and a positive integer k	
Question: Can we find a set $F \subseteq V(G) \times V(G)$ of size at most k such that $\mathsf{rank}(A_{G'}) \leq r$, where $G' = (V(G), E(G) \triangle F)$?	

The set F denotes a set of edits to be performed on the graph G. The $\triangle$ operation acts on the sets as follows: if $(u, v) \in F$ and $(u, v) \in E(G)$ then we delete the corresponding edge from G and if $(u, v) \in F$ and $(u, v) \notin E(G)$ then we add the corresponding edge to G. We also consider a variant of r-Rank Editing, called r-Rank Edge Deletion, where we are allowed to only delete edges.

These problems are also related to some well known problems in graph algorithms. Observe that if $\mathsf{rank}(A_G) = 0$, then G is an empty graph and if $\mathsf{rank}(A_G) = 2$ then G is a complete bipartite graph with some isolated vertices. There are no graphs such that $\mathsf{rank}(A_G) = 1$. If indeed there was a graph such that $\mathsf{rank}(A_G) = 1$, then each of the row of A_G must be either a non-zero vector $\vec{z}$ or an all zero vector. Assume w.l.o.g that $\vec{z}$ is the first row of A_G. The first entry of $\vec{z}$ cannot be one as the diagonal entries are zero in A_G. Hence, the first entry of $\vec{z}$ must be zero. As the matrix A_G is symmetric and the vector $\vec{z}$ is not all zeros, there is a row which is neither all zero nor is it equal to $\vec{z}$. This implies that the rank of A_G must be at least 2, a contradiction. Observe that, for $r = 0$, r-Rank Vertex Deletion is the well known Vertex Cover problem. Similarly for $r = 2$, a solution to r-Rank Edge Deletion is a complement of a solution to Maximum Edge Biclique, where the goal is to find a bi-clique subgraph of the given graph with maximum number of edges [87]. A graph G is called a *Cluster Graph* if every component is a clique. We may also view the above problems as variants of the d-Cluster Vertex Deletion and d-Cluster Editing. In d-Cluster Vertex Deletion, we wish to delete at most k vertices of the graph, so that the resulting graph is a cluster graph with at most d components. Similarly in d-Cluster Editing we wish to add/delete at most k edges to the graph, so that the resulting

graph is a cluster graph with at most d components. These problems are known to be NP-hard and they admit FPT algorithms parameterized by the solution size [39, 51, 57]. In our problems, instead of reducing the graph to a disjoint union of d cliques, we ask that the graph be reduced to a graph of low rank adjacency matrix. In this chapter, we obtain following results about these problems.

- We first show that all the three problems are NP-Complete.
- Then we show that these problems are FPT by designing an algorithm with running time $2^{O(k \log r)} n^{O(1)}$ for r-Rank Vertex Deletion, and an algorithm for r-Rank Edge Deletion and r-Rank Editing running in time $2^{O(f(r)\sqrt{k} \log k)} n^{O(1)}$. Observe that the edge-editing problem r-Rank Editing admits a sub-exponential FPT algorithm.
- Finally, we design polynomial kernels for these problems.

Our results are based on structural observations on graphs with low rank adjacency matrix and applications of some elementary methods in parameterized complexity.

2.2 Preliminaries

We use $\mathbb{R}$ and $\mathbb{N}$ to denote the set of reals and integers, respectively.

Linear Algebra

A vector v of length n is an n-tuple with values from $\mathbb{R}$. A collection of vectors $\{v_1, v_2, \ldots, v_k\}$ are said to be linearly dependent if there exist values $a_1, a_2, \ldots, a_k$; not all zeros, such that $\sum_{i=1}^{k} a_i v_i = 0$. Otherwise these vectors are called linearly independent.

A matrix A of dimension $n \times m$ is a collection of real numbers (a_{ij}) arranged in an $n \times m$ grid. The i-th row of A is defined as the vector $(a_{i1}, a_{i2}, \ldots, a_{im})$ and the j-th column

of A is defined as the vector $(a_{1j}, a_{2j}, \ldots, a_{nj})$. The row set and the column set of A are denoted by $\mathbf{R}(A)$ and $\mathbf{C}(A)$ respectively. For $I \subseteq \mathbf{R}(A)$ and $J \subseteq \mathbf{C}(A)$, we define $A_{I,J} = (a_{ij} \mid i \in I,\ j \in J)$, i.e. it is the submatrix (or minor) of A with the row set I and the column set J. The *rank* of a matrix is the cardinality of the maximum sized collection of columns which are linearly independent. Equivalently, the rank of a matrix is the cardinality of the maximum sized collection of rows which are linearly independent. It is denoted by $\mathsf{rank}(A)$.

Graph Notations and Definitions

For a graph G, we use $V(G)$ and $E(G) \subseteq V(G) \times V(G)$ (a symmetric subset) to denote the vertex set and the edge set of G. For a set of vertices V and a set of edges E on V, we use (V, E) to denote the graph formed by them. Let the vertex set be ordered as $V(G) = \{v_1, v_2, \ldots, v_n\}$. We only consider simple undirected graphs in this chapter, i.e. every edge is distinct and the two endpoints of an edge are also distinct. As we use only undirected graphs, whenever we refer to an edge $(u, v) \in E(G)$ we mean it to refer to the pair $(u, v), (v, u) \in E(G)$.

The *adjacency matrix* of G, denoted by A_G, is defined as the $n \times n$ matrix with values from $\{0, 1\}$ whose rows and columns are indexed by $V(G)$, such that $A_G(i, j) = 1$ if and only if $(v_i, v_j) \in E(G)$. For a vertex $v \in V(G)$, we use $R(v)$ to denote the row of A_G corresponding to v. Similarly we define $C(v)$ to denote the column of A_G corresponding to v. For a set of vertices $S \subseteq V(G)$, we use $R(X)$ and $C(Y)$ to denote the set of rows and columns corresponding to the vertices in S. Similarly for $X, Y \subseteq V(G)$, we use $A_{X,Y}$ to denote the submatrix of A_G corresponding to rows in $R(X)$ and columns in $C(Y)$.

For a vertex $v \in G$, $N(v) = \{u \in G | (u, v) \in E(G)\}$ denotes the neighbourhood of v, and $N[v] = N(v) \cup \{v\}$ denotes the *closed neighbourhood* of v. The complement of a graph G is defined as the graph $\overline{G} = (V(G), \overline{E(G)})$ where $\overline{E(G)} = \{(u, v) | (u, v) \in (V(G) \times V(G)) \setminus$

$E(G), u \neq v\}$. An *independent set* in a graph G is a set of vertices X such that for every pair of vertices $u, v \in X$, $(u, v) \notin E$. A *clique* on n vertices, denoted by K_n, is a graph on n vertices such that every pair of vertices has an edge between them. A *bi-clique* is a graph G, where $V(G) = V_1 \uplus V_2$ and for every pair of vertices $v_1 \in V_1$ and $v_2 \in V_2$, there is an edge $(v_1, v_2) \in E(G)$. When $|V_1| = |V_2| = n$ we denote the biclique by $K_{n,n}$. We can similarly define complete multipartite graphs. For a set of vertices U and a graph G, $G[U]$ denotes the induced subgraph of G on $U \cap V(G)$, and $G \setminus U$ denotes the graph $G[V(G) \setminus U]$.

2.2.1 Some useful results

Given a graph G, we define a relation $\sim$ on $V(G)$. Two vertices u and v are $u \sim v$ if and only if $N(u) = N(v)$. Let A_G denote the adjacency matrix of G. This definition gives us the following lemma.

Lemma 2. *(i) $u \sim v$, if and only if the u^{th} and v^{th} column of A_G are equal.*

(ii) $\sim$ partitions V as $V_1, V_2, \ldots, V_l$ where each V_i is an independent set in G.

(iii) For each pair of V_i and V_j, either there are no edges between V_i and V_j, or $G[V_i \uplus V_j]$ is a bi-clique.

The subsets $V_1, \ldots, V_l$ of V are referred to as the *independent modules* of the graph G. We also refer to V_i as an equivalence class in G. The reduced graph of G is the graph $G^{\sim}$ defined as follows. The vertex set is $V(G^{\sim}) = \{u_1, \ldots, u_l\}$ where l is the number of independent modules of G. The edge set is $E(G^{\sim}) = \{(u_i, u_j) | G[V_i \uplus V_j]$ is a bi-clique $\}$.

The adjacency matrix of $G^{\sim}$ is denoted by $A_G^{\sim}$. Every vertex of $G^{\sim}$ has a distinct neighbourhood which implies that all the columns (rows) of $A_G^{\sim}$ are distinct. We now have the following lemma.

Lemma 3 (Proposition 1. [2])**.** $\mathsf{rank}(A_G) = \mathsf{rank}(A_G^{\sim})$.

The following result by Lovaśz and Kotlov [64] gives us a bound on the number of vertices in $G^{\sim}$, when $\mathsf{rank}(A_G) = r$.

Theorem 2 (see Lovaśz and Kotlov [64])**.** *If the rank of the adjacency matrix of a graph G is r, then the number of vertices in $G^{\sim}$ is at most $c \cdot 2^{\frac{r}{2}}$ for an absolute constant c.*

The following corollary of the above theorem is used extensively in our algorithms.

Theorem 3. *For every fixed r, the number of distinct reduced graphs $G^{\sim}$ such that $\mathsf{rank}(G^{\sim}) = r$, is upper bounded by $c \cdot 2^{2^r}$, for some absolute constant c.*

Proof. Since any such graph has at most $c \cdot 2^{\frac{r}{2}}$ vertices which have an edge incident on it, therefore the number of possible edges is $c^2 \cdot 2^r$. So, the number of such graphs is upper bounded by $2^{c^2} \cdot 2^{2^r}$. □

We will be using the following lemma as well.

Lemma 4 (see Proposition 7 in [2])**.** *The only reduced graph of rank r with no isolated vertices and K_r as a subgraph is K_r itself.*

We also need the following results in the subsequent proofs.

Lemma 5. *$G^{\sim}$ is isomorphic to an induced subgraph of G.*

Proof. Let the vertex set of G and $G^{\sim}$ be as defined above. Consider the map ϕ from $V(G^{\sim})$ to $V(G)$ defined as $\phi(u_i) = v_i$, where v_i belongs to the equivalence class V_i of G corresponding to the vertex u_i. Let $W \subseteq V(G)$ be the images of ϕ, and consider the subgraph $G[W]$. Then observe that for every edge $(u_i, u_j) \in E(G)$ there is an edge $(v_i, v_j) \in E(G[W])$, and similarly for non-edges. Therefore $G^{\sim}$ is isomorphic to $G[W]$. □

Observation 1. *If G' is an induced subgraph of G, then $\mathsf{rank}(A_{G'}) \leq \mathsf{rank}(A_G)$.*

Proof. This follows from the fact that $A_{G'}$ is a submatrix of A_G. □

The following lemma provides an operation to create new graphs preserving the rank.

Lemma 6. *For a graph G let $v \in V(G)$ be a vertex. Let G' be the graph obtained by adding a new vertex to G which has the same adjacency as the vertex v. Then* $\mathsf{rank}(A_G) = \mathsf{rank}(A_{G'})$.

Proof. Adding a copy of v in G corresponds to adding a repeated column and a repeated row in the adjacency matrix of G. Thus the rank is preserved. □

Observation 2. *Let G_1 and G_2 be two graphs and let G' be the disjoint union of G_1 and G_2. Then* $\mathsf{rank}(A_{G'}) = \mathsf{rank}(A_{G_1}) + \mathsf{rank}(A_{G_2})$.

2.3 Reducing Rank by Deleting Vertices

This section considers the problem r-Rank Vertex Deletion.

A solution set is a set of vertices whose deletion reduces the rank of the adjacency matrix of a graph to at most r. We call the given integer r the *target rank*. We start with a structural observation on the solution set of a given instance of r-Rank Vertex Deletion.

Lemma 7. *Let (G, k) be an instance to r-Rank Vertex Deletion and $\sim$ be the equivalence relation on $V(G)$. If $S \subseteq V(G)$ is a minimal solution for the instance (G, k), then it either contains all the vertices of an equivalence class defined by $\sim$ on $V(G)$ or none of it.*

Proof. Let $S \subseteq V(G)$ be a minimal solution containing at least one vertex of an equivalence class V_1 but not containing all the vertices in V_1. All the vertices of $V_1 \setminus S$ have the same neighbourhood in $G \setminus S$, and hence they are contained in the same equivalence class of $G \setminus S$. Let $S' = S \setminus V_1$, we claim that S' is a valid solution. Observe that $G \setminus S'$ can be obtained from $G \setminus S$ by adding $|S \setminus S'|$ copies of any vertex in V_1. By Lemma 6 $\mathsf{rank}(A_{G\setminus S}) = \mathsf{rank}(A_{G\setminus S'})$.

So $S' \subsetneq S$ is also a solution which contradicts the minimality of S, implying that no such S exists. □

The lemma above gives us the following corollary which will be used to get a kernel for the problem.

Corollary 1. *A minimal solution to an instance* (G, k) *of* r-Rank Vertex Deletion *is disjoint from any equivalence class of* G *which has cardinality greater than* k.

The following observation tells us about the solution set of an induced subraph of G.

Observation 3. *Let* S *be a solution to an instance* (G, k) *of* r-Rank Vertex Deletion. *Let* $U \subseteq V(G)$. *Then* $S \cap U$ *is a solution to the instance* $(G[U], k)$.

Proof. Since $G[U] \setminus S$ is an induced subgraph of $G \setminus S$, therefore by Observation 1, $\mathsf{rank}(A_{G[U]\setminus S}) \leq \mathsf{rank}(A_{G\setminus S})$. □

2.3.1 NP-Completeness

Recall that r-Rank Vertex Deletion may be defined as a node-deletion problem with respect to the graph class Π_r. Since Π_r is non-trivial and hereditary, therefore this problem is NP-Complete [70, 101]. We provide the proof for it to show how the Vertex Cover is related to this problem.

Theorem 4. r-Rank Vertex Deletion *is* NP-Complete.

Proof. Let (G, k) be an instance of Vertex Cover. Let H be the complete r-partite graph $K_{k+1,\ldots,k+1}$, a graph having exactly $k+1$ vertices in each partition. The $\sim$-reduced graph of H is K_r, a clique on r vertices and each equivalence class has $k+1$ vertices in it. By Lemma 4, the rank of adjacency matrix of H is r. Let G' be the graph which is the disjoint union of G and H. We claim that Vertex Cover (G, k) is yes if and only if r-Rank Vertex Deletion (G', k) is yes.

In the forward direction, if Vertex Cover (G, k) is a yes instance having S as a solution, then the graph $G'[V \setminus S]$ consists of disjoint union of isolated vertices and H. By Observation 2, $\mathsf{rank}(G'[V \setminus S]) = r$, therefore S is a solution of r-Rank Vertex Deletion (G', k).

For the backward implication, let S be a solution of r-Rank Vertex Deletion (G', k), by Corollary 1, S does not contain any vertex from H. We claim that S is a vertex of G. Suppose not, then there is at least one edge not covered by S, and therefore the rank of $G[V \setminus S]$ is strictly greater than zero. Thus, the rank of $G' \setminus S$, which is the sum of rank of $G[V \setminus S]$ and rank of H, is strictly more than r, a contradiction. □

2.3.2 A parameterized algorithm for r-Rank Vertex Deletion

In this section, we give a parameterized algorithm and a kernel for r-Rank Vertex Deletion. Let (G, k) be an input instance of r-Rank Vertex Deletion. Let $G^{\sim}$ be the reduced graph of G. We have the following corollary of Lemma 7.

Corollary 2. *Let G' be obtained from G by removing all but $k + 1$ vertices from each equivalence class of vertices in G. Then the instances (G, k) and (G', k) are equivalent instances of r-Rank Vertex Deletion.*

Proof. In the forward direction, let S be a solution to the instance (G, k). As G' is an induced subgraph of G, by Observation 1, $\mathsf{rank}(A_{G'\setminus S}) \leq \mathsf{rank}(A_{G\setminus S})$. So $S \cap V(G')$ is a solution to the instance (G', k).

Conversely, let S' be a solution to the instance (G', k). For contradiction assume that S' is not a solution to (G, k), therefore $\mathsf{rank}(A_{G\setminus S'}) > r$. By Corollary 1 S' is disjoint from any equivalence class having at least $k + 1$ vertices in G'. Let $D = V(G') \setminus V(G)$ be the set of deleted vertices. We add deleted vertices one by one in G'. Observe that the reduced graph after addition is the same as reduced graph of G'. By Lemma 6 $\mathsf{rank}(A_{G'\setminus S'}) = \mathsf{rank}(A_{G\setminus S'}) > r$, which is a contradiction.

Therefore the two instances are equivalent. □

The rank of a matrix can be defined alternatively in terms of determinant of its square submatrices using the following well known theorem.

Theorem 5. *A matrix A over real numbers has rank at most r if and only if all the $(r+1) \times (r+1)$ submatrices of A have determinant zero.*

Let $\mathcal{H}$ be a collection of subsets of a set U. A set $S \subseteq U$ is called a *hitting set* of $\mathcal{H}$ if S has non-empty intersection with every set in the collection $\mathcal{H}$. We shall use the notion of hitting set to show that r-Rank Vertex Deletion admits a polynomial kernel and a parameterized algorithm. Let G be a graph and let A be its adjacency matrix. For a graph G and a positive integer r, we define a family of sets $\mathcal{H}_r(G)$ as follows. Let

$$\mathcal{H}_r(G) = \{X \cup Y | X, Y \subseteq V(G), |X| = |Y| = r+1, \mathsf{rank}(A_{X,Y}) = r+1\} \tag{2.1}$$

be a family of sets over $V(G)$.

Lemma 8. *For any $S \subseteq V(G)$, the rank of the adjacency matrix of $G \setminus S$ is at most r if and only if S is a hitting set of $\mathcal{H}_r(G)$.*

Proof. Let A be the adjacency matrix of G and let $\mathcal{H} = \mathcal{H}_r(G)$. For a set $S \subseteq V(G)$, let $A \setminus S$ denote the adjacency matrix of the graph $G \setminus S$. Note that $A \setminus S$ denotes the matrix obtained from A by deleting the rows and columns corresponding to the vertices in S.

Let $S \subseteq V(G)$ be a set such that $A \setminus S$ has rank at most r. Assume that S is not a hitting set of $\mathcal{H}$. Then there is a set in $\mathcal{H}$ which is not hit by S and corresponds to a set of rows and columns whose submatrix has rank equal to $r+1$, which is present in $A \setminus S$. This is a contradiction.

Conversely, let S be any hitting set of $\mathcal{H}$. Assume that $A \setminus S$ has rank greater than $r+1$. Then there is a submatrix of $A \setminus S$ of dimensions $(r+1) \times (r+1)$ which has rank equal

to $r+1$, which is also a submatrix of A. By the definition of $\mathcal{H}$, the set of vertices corresponding to this submatrix is present in $\mathcal{H}$. But S hits this set, which contradicts the fact that the rows and columns corresponding to these vertices are present in $A \setminus S$. □

The Hitting Set problem is defined as follows.

Hitting Set	**Parameter:** k
Input: A set family $\mathcal{F}$ over a universe set U and a positive integer k.	
Question: Does there exist a set $S \subseteq U$ of cardinality at most k such that S is a hitting set of $\mathcal{F}$?	

By Lemma 8, an instance (G, k) is a yes instance of r-Rank Vertex Deletion if and only if $(\mathcal{H}_r(G), k)$ is a yes instance of Hitting Set. This gives us an FPT algorithm for r-Rank Vertex Deletion by branching on any set in $\mathcal{H}_r(G)$.

Theorem 6. *r-Rank Vertex Deletion admits an FPT algorithm running in time $2^{O(k \log r)} n^{O(1)}$.*

Proof. Let (G, k) be an instance of r-Rank Vertex Deletion and let $(\mathcal{H}_r(G), k)$ be an instance of Hitting Set. By Lemma 8, we know that a solution must contain at least one element of each set in $\mathcal{H}_r(G)$. Pick an arbitrary element $S \in \mathcal{H}_r(G)$, for each vertex $v \in S$ recursively solve the sub-problem $(\{U | U \in \mathcal{H}_r(G), v \notin U\}, k-1)$ and add v to the solution. We continue with this branching algorithm until we reach a stage where, either $k \geq 0$ and $\mathcal{H}_r(G)$ is empty and so we return the empty set, or $k = 0$ and $\mathcal{H}_r(G)$ is non empty and so no solution exists for this sub-problem. The branching tree of this algorithm is of depth at most k and at each step it creates at most $2r+3$ sub problems, which gives us the claimed running time.

Observe that if the instance has a solution of size k, then there is always a choice of v in each branching step which leads to an empty sub-problem. Conversely, any solution found by the branching algorithm is a solution to the instance by construction. □

Now we show that r-Rank Vertex Deletion admits a polynomial kernel by an application

of the well known Sunflower lemma. We begin with the definition of a sunflower. A sunflower with k petals and a core Y is a collection of sets $S_1, \ldots, S_k$ such that $S_i \cap S_j = Y$ for all $i \neq j$; the sets $S_i \setminus Y$ are called petals, and we require that none of the sets S_i is empty. Note that a family of pairwise disjoint sets is also a sunflower.

Lemma 9 (Sunflower Lemma [61])**.** *Let $\mathcal{F}$ be a family of sets with each set having cardinality equal to s. If $|\mathcal{F}| > s!(k+1)^s$, then $\mathcal{F}$ contains a sunflower with $k+2$ petals.*

Theorem 7. *r-Rank Vertex Deletion admits a kernel having at most $(2(r+1)) \cdot (2(r+1))!(k+1)^{2r+2}$ vertices.*

Proof. For any fixed r, the collection $\mathcal{H} = \mathcal{H}_r(G)$ can be enumerated in $n^{O(r)}$ time. The set $\mathcal{H}$ is not a uniform family. The cardinality of each set in $\mathcal{H}$ is a number between $r+1$ and $2(r+1)$. We make the family uniform by padding each set in $\mathcal{H}$ with distinct new elements increasing the cardinality to $2(r+1)$. These 'new' elements occur exactly once in each set and the sets contain at least $r+1$ 'old' elements. Any new element can hit exactly one set of the family. Suppose some solution picks a 'new' element, one can replace it by any other 'old' element in the set. The new elements have no other purpose than to make the set family uniform.

If $|\mathcal{H}| > (2(r+1))!(k+1)^{2r+2}$, by Lemma 9 , there is a sunflower in $\mathcal{H}$ with $k+2$ petals. Any hitting set of size k must hit each of the $k+2$ sets present in the sunflower, therefore it must pick an element from the core of the sunflower. Moreover any core will not contain any of the new elements. If the core is empty, then there is no hitting set of size k for $\mathcal{H}$ and we return no. Otherwise, delete one of the sets of the sunflower. We repeat this process until we are left with at most $(2(r+1))!(k+1)^{2r+2}$ sets. Let $\mathcal{H}'$ be a sub-collection of $\mathcal{H}$, such that all the sets in $\mathcal{H} \setminus \mathcal{H}'$ may be safely ignored and all the new elements have been deleted. Observe that $\mathcal{H}$ has a hitting set of size k if and only if $\mathcal{H}'$ has a hitting set of size k. Since each set in $\mathcal{H}'$ contains at most $2(r+1)$ vertices there are at most $2(r+1)(2(r+1))!(k+1)^{2r+2}$ vertices corresponding to $\mathcal{H}'$. Let G' be the graph induced on the vertices corresponding to $\mathcal{H}'$. Observe that $\mathcal{H}'$ is a subset of the collection $\mathcal{H}_r(G')$.

Any solution S to the instance (G', k) of r-Rank Vertex Deletion is a hitting set of of $\mathcal{H}_r(G')$ having size at most k, which is also a hitting set of $\mathcal{H}'$. Any hitting set of $\mathcal{H}'$ is also a hitting set of $\mathcal{H}$, therefore by Lemma 8 S is a solution to the instance (G, k). Conversely, since G' is an induced subgraph of G, by Observation 3, any solution to (G, k) is a solution to (G', k). We output G' as the kernel. □

2.4 Reducing rank by editing edges

This section considers the problem of reducing the rank of a given graph by editing its edge set. As before, we designate the solution size k as the parameter of the problem. For ease of presentation we define the following notation. Let G be a given graph and F be a set of edits performed on G. Let $G' = G \triangle F$ be the edited graph and $H = G'^{\sim}$ be its reduced graph. Each vertex h of H corresponds to an equivalence class V'_h of G'. Let ϕ be a map from $V(G)$ to $V(H)$ which maps a vertex $v \in V(G)$ to the vertex $h \in V(H)$, if $v \in V'_h$. Observe that for a given graph G, F uniquely determines (ϕ, H).

Suppose we are given a pair (ϕ, H) such that for every vertex $h \in V(H)$, there is a vertex $v \in V(G)$ satisfying $\phi(v) = h$; then we can find a unique set F of edits to $E(G)$ such that the reduced graph of H is the same as the reduced graph $G \triangle F$. We now show how to construct F given a pair (ϕ, H). We start with an empty set F and add edges to it as follows. For each pair of vertices u, v in G, if $(u, v) \in E(G)$ and $(\phi(u), \phi(v)) \notin E(H)$, then we need to delete the edge (u, v); therefore we add (u, v) to F. Similarly, if $(u, v) \notin E(G)$ and $(\phi(u), \phi(v)) \in E(H)$, then we add the edge (u, v) to F. This completes the description of F and we see that it is uniquely determined by ϕ and H.

We start with the following lemma about the structure of minimal solutions of r-Rank Editing.

Lemma 10. *Let F be a minimal solution to an instance (G, k) of r-Rank Editing. For any two independent equivalence classes V_i and V_j of G, either F contains all the edges*

$V_i \times V_j$ *or it contains none of them.*

Proof. Let $G' = G \triangle F$ denote the graph obtained after performing the edits in F on G. Let $H = G'^{\sim}$ and ϕ be the map as defined above. If F contains some but not all edges from $V_i \times V_j$, then there exists an equivalence class V_1 in G whose vertices go into different equivalence classes of G'. Let u be a vertex in V_1 which received the minimum number of edits among all the vertices in V_1, and let $h_u = \phi(u)$. Pick a vertex $v \in V_1$ which goes into a different equivalence class than that of u in the edited graph G'. Define a new map ψ from $V(G)$ to $V(H)$ as follows. Let $\psi(v) = h_u$ and $\psi(w) = \phi(w)$ for all $w \in V \setminus \{v\}$. Let F'' be the set of edits corresponding to (ψ, H) and $G'' = G \triangle F''$. Let H' be the induced subgraph of H such that every vertex of H' is the image of some vertex of G under the map ϕ. If $V(H') = V(H)$, then $G''^{\sim}$ is the graph H. Otherwise, by Lemma 5, $H'^{\sim}$ is an induced subgraph of H'. Since H' is an induced subgraph of H, therefore $H'^{\sim}$ is isomorphic to an induced subgraph of H. Since $G''^{\sim} = H'^{\sim}$, therefore by Observation 1, $\mathsf{rank}(A_{G''}) \leq \mathsf{rank}(A_H) = \mathsf{rank}(A_{G'})$. Let F'_u be the set of edits in F' with u as one endpoint, and let $F''_v = \{(v, w) | (u, v) \in F'_u, w \neq v\}$. Observe that $F'' = (F' \cup F''_v) \setminus (F'_v \cup \{(u, v)\})$, where F'_v denotes the edits in F' with v as one end point. Then clearly $|F''| \leq |F'|$.

We iteratively perform the above operation for all those vertices in V_1 which are mapped to a different equivalence class in G'. Thus, we can find a solution such that all the vertices of V_1 go to the same equivalence class after editing. Moreover, applying this operation ensures that all the vertices in an equivalence class in G receive the same set of edits.

To complete the proof, observe that the graph induced on two equivalence classes is a complete bipartite graph. After editing, suppose V_i and V_j are contained in W_i and W_j respectively, where W_i and W_j are the equivalence classes of the edited graph. If $W_i = W_j$, then all the vertices between V_i and V_j have been deleted. If $W_i \neq W_j$, then all the edges of $V_i \times V_j$ are present if and only if W_i and W_j have an edge. □

This gives us the following corrollary.

Corollary 3. *If F is a minimal solution to an instance (G, k) of r-Rank Editing, then F does not affect any equivalence class of G having at least $k + 1$ vertices.*

The results in this section also holds for r-Rank Edge Deletion as deletion of edges is a special case of edge editing. In particular, for a graph G, if one restricts the set of edits F to be a subset of $E(G)$, then $G \triangle F$ corresponds to the graph obtained after deleting edges in F from G.

2.4.1 NP-Completeness

We give a reduction from the d-Cluster Editing which is known to be NP-Complete for any $d \geq 2$ [96]. Additionally, this probem is FPT when parameterized by k [39]. The input for d-Cluster Editing consists of a graph G and two positive integers d and k. The problem is to find a set F of k edges such that the graph $(V, E \triangle F)$ is the partition of at most d disjoint cliques.

Theorem 8. *r-Rank Editing is NP-Complete for any $r \geq 3$*

Proof. The problem is in NP as one can verify a claimed solution in polynomial time.

We reduce 2-Cluster Editing to r-Rank Editing to show that the problem is NP-Hard for a fixed $r \geq 3$. To simplify notation, let (G, k) be the given instance, $V = V(G)$, $E = E(G)$, $\overline{E} = E(\overline{G})$ and $n = |V|$. An input instance for r-Rank Editing is created as follows. Let Z be the complete $(r - 2)$-partite graph $K_{k+2,\ldots,k+2}$, each partition has exactly $k + 2$ vertices and all the edges between any two partitions are present. By Lemma 4 it has rank $r - 2$ as it has K_{r-2} as its reduced graph. We construct a new graph H by taking the complement $\overline{G}$ of G, a copy of the graph Z and adding all the edges between the vertices of Z and vertices in $\overline{G}$. Then the input instance of r-Rank Editing is (H, k).

In the forward direction, suppose that the instance (G, k) has a solution F of size at most k such that $G' = (V, E \triangle F)$ is a graph consisting of at most two disjoint cliques. Thus

$\overline{G}'$ consists of either isolated vertices or a complete bi-partite graph. Therefore $H' = \left(V \cup V(Z), (\overline{E} \triangle F) \cup E(Z) \cup E(Z, V(G))\right)$ is a graph with either K_r or K_{r-1} as its reduced graph and has rank at most r.

In the reverse direction, suppose F is a minimal solution of size at most k for (H, k). Let $H' = H \triangle F$ be the edited graph. Each equivalence class of $H[Z]$ has $k+2$ vertices and $|F| \leq k$, therefore by Corollary 3, F is disjoint from all the edges incident on the vertices of Z i.e. $F \cap \{(E(Z) \cup E(Z, V(G))\} = \phi$. This means that only the edges between the vertices of G can be edited implying $F \subseteq V(G) \times V(G)$ and $H' = \left(V \cup V(Z), (\overline{E} \triangle F) \cup E(Z) \cup E(Z, V(G))\right)$. Hence, any equivalence class of H' is contained entirely in either $V(G)$ or $V(Z)$. Let X and Y be two equivalence classes of H' which are contained in $V(G)$, then $H'[X \cup Y]$ is a bi-clique. Then observe, that the induced subgraph $H'[X \cup Y \cup V(Z)]$ has no isolated vertices and it has K_r as it's reduced subgraph. Moreover there are no isolated vertices in H', as every vertex has at least $k+1$ neighbours in H; therefore, by Lemma 4, the reduced graph of H' is also K_r, which implies that X and Y form a partition of $V(G)$. If we cannot find such an X and Y, then $H'[V(G)]$ contains no edges and the reduced graph of H' is K_{r-1}. Therefore, $H'[V(G)]$ is either an independent set or a bi-clique and $\overline{H'[V(G)]}$ can be partitioned into at most 2 cliques. As $(V, E \triangle F) = \overline{H'[V(G)]}$, we have that F is a solution to the instance (G, k). □

To prove the r-Rank Edge Deletion problem is NP-Complete, we give a reduction from the Maximum Edge Bi-Clique Problem [87].

Theorem 9. *For any fixed positive integer $r \geq 2$, r-Rank Edge Deletion is* NP-Complete.

Proof. We give a reduction from the Maximum Biclique Problem, which is defined as follows: Given a graph G and an integer K, find a bi-clique (A, B) in G such that $|A| \times |B| \geq K$. This problem was shown to be NP-Complete by Peeters [87].

Given an instance (G, K) of Maximum bi-clique Problem, we define an instance of 2-Rank Edge Deletion as $(G, m - K)$, where m is the number of edges in G. If (G, K) is a yes

instance, then deleting all the edges other than the edges of the bi-clique gives a graph of rank 2. Conversely, if 2-Rank Edge Deletion$(G, m - K)$ is a yes instance, then the edges remaining after deleting the solution form the required bi-clique.

Similar to the proof of NP-Completeness for r-Rank Vertex Deletion, we can generalize the reduction above for any $r \geq 2$ □

2.4.2 A parameterized algorithm for r-Rank Editing

In this section, we show that r-Rank Editing is FPT parameterized by the solution size and admits a polynomial kernel. The results and algorithms for r-Rank Edge Deletion follow in a similar manner using the same techniques. A vertex $v \in V$ is called an *affected vertex* with respect to the solution F if there is some edge in F which is incident on v. By Lemma 10, if a vertex in an equivalence class V_1 is affected with respect to a minimal solution F, then every vertex in V_1 is also affected. In that case, we say that the equivalence class V_1 is affected by F.

Lemma 11. *Let (G, k) be an instance of the r-Rank Editing problem. If $G^{\sim}$ has more than $c \cdot 2^{r/2} + 2k$ vertices, then it is a no instance.*

Proof. Let (G, k) be a yes instance and G' be the graph obtained from G after k edits. Since the graph G' obtained after editing has rank at most r, by Theorem 3 it has at most $c \cdot 2^{r/2}$ equivalence classes. By Lemma 10, any minimal edit affects all the edges between two equivalence classes, therefore there are at most $2k$ equivalence classes of G which are affected. Observe that, if V_1 and V_2 are two distinct equivalence classes of G, then they have distinct neighbourhoods. If they end up in the same equivalence class of G', then at least one of them must be affected. Every equivalence class of G ends up in some equivalence class of G' which may have at most $c \cdot 2^{r/2}$ equivalence classes, therefore there can be at most $c \cdot 2^{r/2} + 2k$ equivalence classes in G. □

Using the result above and Corollary 3, we have the following kernel for r-Rank Editing.

Theorem 10. *r-Rank Editing admits a kernel with $O((2^{r/2} + 2k) \cdot (k + 1))$ vertices.*

Proof. If (G, k) has more than $c \cdot 2^{r/2} + 2k$ equivalence classes then, by Lemma 11, there is no solution for this instance. Otherwise, consider the graph G' obtained by removing all but $k + 1$ vertices of each equivalence class of G. We will show that the instances (G, k) and (G', k) are equivalent.

In the forward direction, let F be a solution to (G, k). By Corollary 3, F only affects those equivalence classes of G which have at most k vertices. All the vertices of affected equivalence classes are present in G', hence $F \subseteq V(G') \times V(G')$. Therefore $G' \triangle F$, is an induced subgraph of $G \triangle F$ implying it is a solution to (G', k).

Conversely, let F' be a solution to the instance (G', k). Let $D = V(G) \setminus V(G')$ be the set of deleted vertices. For every vertex $w \in D$ there is a vertex $x \in V(G')$ which has the same neighborhood as w, this so because F leaves the neighborhood of x unchanged by Corollary 3. By Lemma 6, adding the vertices in D to $G' \triangle F$ does not change its rank. Hence $\mathsf{rank}(A_{G' \triangle F'}) = \mathsf{rank}(A_{G \triangle F'})$ implying that F' is a solution to the instance (G, k).

We output (G', k) as the kernel for the instance (G, k). Observe that the instance (G', k) has at most $c \cdot 2^{r/2} + 2k$ equivalence classes and each equivalence class has at most $k + 1$ vertices, so $V(G')$ has the claimed size bound. □

In the rest of the section, we would be presenting the algorithm for r-Rank Editing. We can get an algorithm using the techniques introduced so far. Let A be the adjacency matrix of G and let $A_{X,Y}$ be a submatrix of A having dimension $(r + 1) \times (r + 1)$ and rank equal to $r + 1$. Any solution to the instance (G, k) must edit an entry of $A_{X,Y}$. This observation gives us an FPT algorithm for r-Rank Editing which is similar to the algorithm in Theorem 6.

Theorem 11. *r-Rank Editing admits an FPT algorithm running in time $2^{O(k \log r)} n^{O(1)}$.*

However, an asymptotically faster sub-exponential algorithm for r-Rank Editing can be obtained by using an algorithm of Damaschke et. al. [29]. We will reduce r-Rank Editing to a problem called as H-Bag Editing. For that, we next state some definitions and notations used. We begin with the definition of $\approx$, which is a relation defined using the closed neighbourhood of vertices in G. For two vertices $u, v \in V(G)$, we say $u \approx v$ if and only if $N[u] = N[v]$ in G. Observe that $\approx$ is an equivalence relationship on vertices of G and each equivalence class induces a clique in G. Let $V_1, \ldots, V_l$ be a partition of vertices $V(G)$. We define the $\approx$-reduced graph $G^{\approx}$ as follows: the vertex set is $V(G^{\approx}) = \{u_1, \ldots, u_l\}$ and the edge set is $E(G^{\approx}) = \{(u_i, u_j)$ if and only if all the edges between V_i and V_j are present in $E(G)\}$. Note that $G^{\approx}$ may contain isolated vertices.

We next define the H-Bag Editing problem formally.

H-Bag Editing	*Parameter: k*
Input: Graphs G, H and an integer k.	
Question: Can we find a set F of k edges such that $\approx$-reduced graph of $G' = (V(G), E(G) \triangle F)$ is an induced subgraph of H ?	

Theorem 12 ([29])**.** *The H-Bag Editing problem with a fixed graph H can be solved in $O^*(2^{O(\sqrt{k}\log k)})$ time*[2].

We show how to use the theorem above to obtain a sub-exponential FPT algorithm for r-Rank Editing. To accomplish this we need to find how the $\sim$-reduced and $\approx$-reduced graphs are related. This information is provided by the following lemma.

Lemma 12. *Given a graph G, let $G^{\sim}$ denote the reduced graph with respect to the excluded neighborhood relation $\sim$ and let $G^{\approx}$ denote the reduced graph with respect to the closed neighborhood relation $\approx$. Then the graph $(\overline{G})^{\approx}$ is the complement of $G^{\sim}$.*

Proof. It suffices to prove that the equivalence classes of $\approx$ in $\overline{G}$ are exactly the same as the equivalence classes of $\sim$ in G. For $u, v \in V(G)$, suppose $u \sim v$ which implies

[2]The O^* notation hides the terms depending on r which is assumed to be a constant, and polynomial multiplicative factors.

$N_G(u) = N_G(v)$. Therefore $N_{\overline{G}}[u] = V(G) \setminus N_G(u) = V(G) \setminus N_G(v) = N_{\overline{G}}[v]$, moreover $(u, v) \in E[\overline{G}]$, which implies $u \approx v$ in $\overline{G}$. We can show the reverse direction in a similar way. Since the $\sim$-equivalence classes of G are same as the $\approx$-equivalence classes of $\overline{G}$, their corresponding reduced graphs are complements of each other. □

Corollary 4. *If H is a $\sim$-reduced graph, then its complement is a $\approx$-reduced graph.*

Theorem 13. *An instance (G, k) of r-Rank Editing can be solved in $O^*(2^{O(\sqrt{k}\log k)})$ time.*

Proof. For a fixed r, the number of reduced graphs whose adjacency matrix has rank r is a function of r alone by Theorem 3 [2]. We generate all such reduced graphs H, using algorithm in [2](Section 3), and we check if a given graph G can be edited such that its reduced graph is H. If there is one such H, then by Lemma 12, $(\overline{G}, \overline{H}, k)$ is a yes instance of H-Bag Editing and a solution to it can be computed using Theorem 12.

Finally, we note that the solution returned by H-Bag Editing is the solution of r-Rank Editing. Let F be the solution returned by H-Bag Editing, this means that the $\approx$-reduced graph of $\overline{G} \triangle F$ is an induced graph of $\overline{H}$. By Corollary 4, the $\sim$-reduced graph of $G \triangle F$ is an induced graph of H. □

Chapter 3

Rank Reduction of Oriented Graphs by Vertex and Edge Deletions[1]

In this chapter, we continue our study of graph modification problems defined by reducing the rank of the adjacency matrix of the given graph. We extend our results from modifying the rank of adjacency matrix of an undirected graph to modifying the rank of skew-adjacency matrix of oriented graphs. An instance of a graph modification problem takes as input a graph G and a positive integer k, and the objective is either to delete k vertices/edges or to edit k edges so that the resulting graph belongs to a particular family $\mathcal{F}$ of graphs. Given a fixed positive integer r, we define $\mathcal{F}_r$ as the family of oriented graphs where for each $G \in \mathcal{F}_r$, the rank of the skew-adjacency matrix of G is at most r. Using the family $\mathcal{F}_r$ we do algorithmic study, both in classical and parameterized complexity, of the following graph modification problems: r-Rank Vertex Deletion, r-Rank Edge Deletion. We first show that both the problems are NP-Complete. Then, we show that these problems are fixed parameter tractable (FPT) by designing an algorithm with running time $2^{O(k \log r)} n^{O(1)}$ for r-Rank Vertex Deletion, and an algorithm for r-Rank Edge Deletion running in time $2^{O(f(r)\sqrt{k}\log k)} n^{O(1)}$. In addition to our FPT results, we design

[1]This chapter is based on a joint work with Saket Saurabh [81].

polynomial kernels for these problems. Our main structural result, which is the fulcrum of all our algorithmic results, is that for a fixed integer r the size of any "reduced graph" in $\mathcal{F}_r$ is upper bounded by 3^r. This result is of independent interest and generalizes a similar result of Kotlov and Lovász [64] regarding reduced oriented graphs of rank r.

3.1 Introduction

Editing a graph by either deleting vertices or deleting edges or adding edges, such that the resulting graph satisfies certain properties or becomes a member of some well-understood graph class, is one of the basic problems in graph theory and graph algorithms. These problems are called graph modification problems. Most of the graph modification problems are NP-Complete [101, 70] and thus they are subjected to intensive study in algorithmic paradigms that are meant for coping with NP-Completeness [44, 77, 41, 79]. These paradigms among others include applying restrictions on inputs, approximation algorithms and parameterized complexity.

Graph modification problems have been at forefront of research in parameterized complexity and several interesting and important results have been obtained recently. In fact, just over the course of the last couple of years there have been results on parameterized algorithms for Chordal Editing [19], Unit Interval Editing [17], Interval Vertex (Edge) Deletion [20, 18], Proper Interval Completion [10], Interval Completion [11], Chordal Completion [42], Cluster Editing [39], Threshold Editing [35], Chain Editing [35], Trivially Perfect Editing [36, 37] and Split Editing [46]. Even more recently, a theory for lower bounds for these problems has also been proposed [9]. We would like to mention that the above list is not comprehensive but rather illustrative.

To the best of our knowledge, all the articles on graph modification problems in parameterized complexity (except Ladder Graphs which are defined using forbidden topological minors) is about modifying the input graph to a graph in a family $\mathcal{F}$, defined by either

forbidden induced subgraphs or minors. In Chapter 2, we studied graph modification problems on undirected graph defined by reducing the rank of the adjacency matrix of the input graph. Our main goal in this chapter is to continue our study of graph modification problems defined by some algebraic properties like rank of a given adjacency matrix. In particular, we extend our results to oriented graphs (loopless directed graphs with at most one arc between vertices) from undirected graphs in the realm of classical and parameterized complexity.

An oriented graph can be obtained from a simple undirected graph by assigning direction to each of its edges. For an oriented graph D the spectral properties of its skew-symmetric matrix A_D have been studied in the past by Cavers et al [21]. Bicyclic oriented graphs with skew-rank 2 or 4 have also been characterized by the rank of their skew-adjacency matrix by Qu and Yu [88]. This chapter studies the problem of modifying the rank of the skew-adjacency matrix. In particular, given a fixed positive integer r, we define $\mathcal{F}_r$ as the family of oriented graphs where for each $G \in \mathcal{F}_r$, the rank of the skew-adjacency matrix of G is at most r. We study the following problems.

r-Rank Vertex Deletion	**Parameter:** k
Input: An oriented graph D and a positive integer k	
Question: Does there exist a set $S \subseteq V(D)$ of size at most k such that $\mathsf{rank}(A_{D\setminus S}) \leq r$?	

r-Rank Edge Deletion	**Parameter:** k
Input: An oriented graph D and a positive integer k	
Question: Does there exist a set $F \subseteq E(D)$ of size at most k such that $\mathsf{rank}(A_{D'}) \leq r$, where $D' = (V(D), E(D) \setminus F)$?	

These problems are also related to some well known problems in graph algorithms. If $\mathsf{rank}(A_D) = 0$, then D is a disjoint union of isolated vertices. So for $r = 0$, r-Rank Vertex Deletion is the well known Vertex Cover problem. Similarly for $r = 2$, after converting an undirected graph to an oriented graph, a solution to r-Rank Edge Deletion

is the complement of a solution to the Maximum Edge Biclique problem. In the Maximum Edge Biclique problem the goal is to find a complete bipartite subgraph of the given graph with maximum number of edges [87] possible. These problems are also related to the concept of "rigidity of matrices" [78].

Our Results and Methods. In this chapter, we obtain the following results about these problems.

1. We first show that both the problems are NP-Complete.

2. Then we show that these problems are FPT by designing an algorithm with running time $2^{O(k \log r)} n^{O(1)}$ for r-Rank Vertex Deletion, and an algorithm for r-Rank Edge Deletion running in time $2^{O(f(r)\sqrt{k} \log k)} n^{O(1)}$. Note that the edge deletion problem admits a sub-exponential FPT algorithm.

3. Finally, we design polynomial kernels for these problems.

The main difficulty in extending our result from undirected to oriented graphs is that the entries of an adjacency matrix of an undirected graph are from $\{0, 1\}$, while the entries of the skew-adjacency matrix of an oriented graph are from $\{0, 1, -1\}$. Also, the adjacency matrix of an undirected graph is symmetric while for an oriented graph it is skew-symmetric. Our algorithmic results are based on a structural result regarding oriented graphs whose adjacency matrices have low rank. In particular, we define an equivalence relation on the neighborhood of oriented graphs and show that the size of oriented graphs, in which each equivalence class has size one, is upper bounded by a function of the rank of its adjacency matrix. In case of undirected graphs, such results were proved by Kotlov and Lovász [64]. In particular, they showed that if a graph is reduced (if no two vertices have the same set of neighbours) and its adjacency matrix has rank r, then its size is upper bounded by $2^{r/2}$. See Akbari et al. [2] for more details. We show that the "reduced" oriented graphs of rank r have size at most 3^r. This result could be of independent interest. We use this result to define a subfamily of $\mathcal{F}_r$ (which is sufficient for our purposes) whose

size is upper bounded by a function of r alone. This together with standard methods in parameterized complexity easily imply FPT and kernel result for r-Rank Vertex Deletion. To obtain sub-exponential algorithm for r-Rank Edge Deletion we do a modification to an algorithm of Damaschke et. al. [29] and use it as a subroutine.

3.2 Preliminaries

We would be reusing the notation from Chapter 2. We next define the new notations which are used in this chapter. The span of a set of vectors $\{v_1, \ldots, v_m\}$, denoted as $\mathsf{span}(v_1, \ldots, v_m)$, is defined as the set $\{a_1v_1 + \cdots + a_mv_m : a_1, \ldots, a_m \in \mathbb{R}\}$. The i-th row of A is the vector $(a_{i1}, a_{i2}, \ldots, a_{im})$ and is denoted using A_i. The j-th column of A is the vector $(a_{1j}, a_{2j}, \ldots, a_{nj})$ and is represented using A^j. We define $A[I, J] = (a_{ij} \ : \ i \in I, \ j \in J)$, i.e. it is the submatrix of A with the row set I and the column set J. When $I = J$, the submatrix $A[I, I]$ is called as a principal submatrix of A.

We use D to denote an oriented graph and its edges will be referred as arcs. In this chapter we will be exclusively concerned with loopless oriented graphs; they have at most one arc between any two vertices i.e. for two vertices $u, v \in V(D)$ only one the following conditions holds; *(i)* $(u, v) \in E(D)$, *(ii)* $(v, u) \in E(D)$ or *(iii)* there is no arc between u and v. For an oriented graph D, given a vertex $v \in V(D)$ we denote the set of vertices with an outgoing arc to v by $N^-(v) = \{u : (u, v) \in E(D)\}$. Similarly, we define the set of vertices with an incoming arc from v as $N^+(v) = \{u : (v, u) \in E(D)\}$. A vertex in D is called as an isolated vertex if $N^+(v) = N^-(v) = 0$. The *skew-adjacency matrix* of D, denoted by A_D, is a $|V(D)| \times |V(D)|$ skew-symmetric matrix ($A = -A^{\mathbf{T}}$) with entries from $\{-1, 0, +1\}$ whose columns and rows are indexed using vertices and $A_D(u, v) = -A_D(v, u) = 1$ if and only if (u, v) is an arc in D. *For simplicity, we will refer to the skew-adjacency matrix of an oriented graph as the adjacency matrix.* A *directed bi-clique* is a directed graph D defined over two disjoint vertex sets V_1 and V_2, where $V(D) = V_1 \cup V_2$ and for every pair of vertices $v_1 \in V_1$ and $v_2 \in V_2$, there is an arc $(v_1, v_2) \in E(D)$. Note that in a

directed bi-clique all the arcs are directed from one part to the other and both V_1 and V_2 are independent sets. Given any two graphs G_1 and G_2 their disjoint union is denoted by $G_1 \uplus G_2$.

For a vertex $v \in V(G)$, we use $R(v)$ to denote the row vector of A_G corresponding to v. Similarly we use $C(v)$ to denote the column vector of A_G corresponding to v. For $X, Y \subseteq V(G)$, we use $A_G[X, Y]$ to denote the submatrix of A_G corresponding to rows in $R(X)$ and columns in $C(Y)$. For a graph G and a set of vertices $U \subseteq V(G)$, $G[U]$ denotes the induced subgraph of G, and $G \setminus U$ denotes the graph $G[V(G) \setminus U]$.

3.3 A structural result on oriented graphs of rank r

In this section we give our main structural result about oriented graphs of low rank. This result will be exploited heavily in all the algorithms presented here. We start by defining an equivalence relation based on the neighborhood of vertices in oriented graphs. Using this we show that the number of vertices in an oriented graph of rank r, where each equivalence class of this relation has size exactly one, is upper bounded by 3^r.

Given an oriented graph D we define a relation $\sim$ on $V(D)$. Two vertices $u, v \in V(D)$ are $u \sim v$ if they have the *same neighborhood*. That is, $u \sim v$ if and only if $N^+(u) = N^+(v)$ and $N^-(u) = N^-(v)$. This equivalence relation is similar to the one defined in the context of undirected graphs in [2]. From the definition of the equivalence relation $\sim$ the following properties regarding it easily follow.

Lemma 13. *Let D be an oriented graph and $\sim$ be the relation as defined above. Then,*

(i) *$u \sim v$ if and only if $R(u) = R(v)$.*

(ii) *$\sim$ partitions $V(D)$ as $V_1, , \ldots, V_\ell$ and each V_i is an independent set in D.*

(iii) *For each pair of distinct V_i and V_j, either there are no edges between V_i and V_j or $D[V_i \cup V_j]$ is a directed bi-clique.*

When the graph is clear from the context, an equivalence class of $\sim$ will be referred to as a *module* or an *equivalence class*. Given an oriented graph D, we define $D^{\sim}$ to be the *reduced graph* of D obtained as follows. Let $V_1, \ldots, V_\ell$ denote the equivalence classes of $\sim$ in D. The vertex set of $D^{\sim}$ is $V(D^{\sim}) = \{V_i : i \in [\ell]\}$. The edge set of $D^{\sim}$ is

$$E(D^{\sim}) = \{(V_i, V_j) : D[V_i \cup V_j] \text{ is a directed bi-clique with all arcs from } V_i \text{ to } V_j\}.$$

For an oriented graph D the operation of obtaining the reduced graph can also be thought of as selecting one vertex from each equivalence class arbitrarily and then constructing the induced subgraph of D over them. This gives us the following.

Observation 4. *$D^{\sim}$ is an induced subgraph of D and the columns of adjacency matrix of $D^{\sim}$ are distinct.*

The operation of obtaining a reduced graph preserves the rank, as the operation simply removes repeated columns.

Lemma 14. $\mathsf{rank}(A_D) = \mathsf{rank}(A_{D^{\sim}})$.

In the rest of the section, we prove that the number of oriented graphs whose adjacency matrix has rank r is bounded by a function of r. Similar result exists for the case of undirected graphs [64]. It is well known that any symmetric matrix A has a principal submatrix of rank equal to $\mathsf{rank}(A)$ (see Theorem 3.9.8 in [38]). Similarly, for a skew-symmetric matrix the following holds.

Lemma 15. *Any skew-symmetric matrix A has a principal submatrix of rank equal to* $\mathsf{rank}(A)$.

As the determinant of an odd sized skew-symmetric matrix is zero we get the following lemma.

Lemma 16 (see Theorem 6 in Section 10.3 of [55])**.** *The rank of a skew-symmetric matrix is an even number.*

Using Lemma 15, along with the observation that for a skew-symmetric matrix B we have $B = -B^T$ we get the following corollary.

Corollary 5. *Any rank r skew-symmetric matrix A, with full rank principal submatrix $B = A[[r], [r]]$, can be written in the following form for an appropriate sized matrix X,*

$$\begin{bmatrix} B & BX \\ X^T B & X^T BX \end{bmatrix}. \tag{3.1}$$

Theorem 14. *Any rank r skew-symmetric matrix A with entries from $\{-1, 0, +1\}$ and pairwise distinct columns has at most 3^r columns.*

Proof. For an $n \times n$ matrix A, let $R = [r]$ and assume without loss of generality that the principal submatrix $B = A[R, R]$ is of full rank. By Corollary 5, matrix A can be written in the form as shown in Equation 3.1. We prove that no two columns in the matrix $A[R, [n]]$ are repeated. As B is of full rank it has no repeated columns. Assume that the columns i and j in BX are repeated. As A is skew-symmetric, the rows i and j are repeated in $X^T B$. Therefore, the product $(X^T B)X$ has rows i and j repeated which again, as A is skew-symmetric, implies that the columns i and j are repeated in $X^T BX$. Hence, the columns i and j are repeated in A. Therefore, it suffices to enforce that no column in $A[R, [n]]$ is repeated. For an upper bound notice that the maximum possible number of distinct vectors of length r with entries from $\{-1, 0, +1\}$ is 3^r. □

As a consequence of the theorem above, we have the following.

Corollary 6. *Let D be an oriented graph having adjacency matrix A of rank r. Then the number of vertices in $D^{\sim}$ is at most 3^r.*

3.4 Reducing Rank by Deleting Vertices

We will be considering r-Rank Vertex Deletion in this section. For a graph D, we denote the graph obtained after deleting a set of vertices $S \subseteq V[D]$ by $D \setminus S$. We begin with some structural lemma about modules of induced graphs.

Lemma 17 (folklore)**.** *Let D be an oriented graph, $S \subseteq V(D)$ and $u, v \in V(D) \setminus S$. If u and v have the same neighborhood in D, then they have the same neighborhood in $D \setminus S$ and* $\mathsf{rank}(A_{D\setminus S}) \leq \mathsf{rank}(A_D)$.

As a corollary of Lemma 14, we have the following.

Corollary 7. *Given an oriented graph D and a vertex $u \in V(D)$, let D' be the graph obtained by adding a new vertex u' in D which has exactly the same neighborhood as u. Then,* $\mathsf{rank}(A_D) = \mathsf{rank}(A_{D'})$.

We will be working with solutions of a special form, we define S to be a *minimal solution* to an instance (D, k) of r-Rank Vertex Deletion if it either contains all the vertices of an equivalence class defined by $\sim$ on $V(D)$ or none of it.

Lemma 18. *Let (D, k) be an input instance of r-Rank Vertex Deletion and $\sim$ be the equivalence relation on $V(D)$. If $S \subseteq V(D)$ is a solution for the instance (D, k), then there exists a* minimal *solution S' such that $S' \subseteq S$.*

Proof. Let $V_1, \ldots, V_t$ be the modules of $\sim$ in D. Assume that V_i is a module such that $V_i \setminus S$ is nonempty and some vertex $v \in V_i$ is contained in the solution S. Pick any vertex $u \in V_i \setminus S$. Denote the graph obtained by deleting a vertex subset $X \subseteq V(D)$ as $D_X = D \setminus X$. Consider the graph $D_{S\setminus\{v\}}$, observe that both u and v are present in $D_{S\setminus\{v\}}$. Moreover, both u and v have the same neighborhood in $D_{S\setminus\{v\}}$ by Lemma 17. Hence, by Corollary 7, $\mathsf{rank}(A_{D_S}) = \mathsf{rank}(A_{D_{S\setminus\{v\}}})$.

Using the argument above, on deleting the vertices in $V_i \cap S$ from S we find that $\mathsf{rank}(A_{D\setminus S}) = \mathsf{rank}(A_{D\setminus(S\setminus V_i)})$. Repeating this procedure for all the modules which are partially contained in S gives a solution of the desired form. □

As an immediate consequence of the lemma above, we have the following.

Corollary 8. *Any solution S for an instance (D, k) of r-Rank Vertex Deletion is disjoint from a $\sim$-module of size strictly more than k.*

3.4.1 Complexity of r-Rank Vertex Deletion

The r-Rank Vertex Deletion problem generalizes Vertex Cover. In this section, for a fixed r we give a reduction from Vertex Cover to r-Rank Vertex Deletion which would prove that no FPT algorithm is possible for r-Rank Vertex Deletion which uses rank alone as the parameter. This is so, because an algorithm parameterized by rank alone would solve Vertex Cover in polynomial time, contradicting the assumption that $\mathsf{P} \neq \mathsf{NP}$.

We first make some observations about the rank of the adjacency matrix of a graph and then proceed to the hardness result. The adjacency matrix of a graph which is the disjoint union of two graphs D_1 and D_2 is of the form

$$A_{D_1 \uplus D_2} = \begin{bmatrix} A_{D_1} & 0 \\ 0 & A_{D_2} \end{bmatrix}. \tag{3.2}$$

Hence, we get the following observation.

Observation 5. *Let D_1 and D_2 be two oriented graphs with disjoint vertex sets. Let $\mathsf{rank}(A_{D_1}) = r_1$ and $\mathsf{rank}(A_{D_2}) = r_2$. Then the rank of adjacency matrix of the oriented graph $D_1 \uplus D_2$ is $r_1 + r_2$.*

The case when the adjacency matrix is a zero matrix, we have the following.

Observation 6. *For a graph D the rank of A_D is zero if and only if D is the disjoint union of isolated vertices.*

Theorem 15. r-Rank Vertex Deletion *is* NP-Complete.

Proof. Let F be a reduced oriented graph with $\mathsf{rank}(A_F) = r$. Obtain the graph F_{k+1} by making $k+1$ copies, keeping the neighborhood same, of each vertex in F, observe that $\mathsf{rank}(A_{F_{k+1}}) = r$ by Corollary 7. Let (G, k) be an input instance of Vertex Cover. The graph G is undirected, we convert it into an oriented graph D by arbitrarily assigning "direction" to each edge in G, i.e. for each pair $(u, v), (v, u) \in E(G)$ we keep only one in $E(D)$. Then, the input instance for the problem r-Rank Vertex Deletion is $(D \uplus F_{k+1}, k)$.

For the forward direction, assume that S is any solution set for Vertex Cover. Then S is also a solution set for r-Rank Vertex Deletion by Observation 5, as $D \setminus S$ is a disjoint union of isolated vertices and $\mathsf{rank}(A_{F_{k+1}}) = r$.

For the backward direction, observe that any solution S of r-Rank Vertex Deletion is disjoint from $V(F_{k+1})$ by Corollary 8. Moreover $D \setminus S$ must consist of isolated vertices, otherwise rank of the adjacency matrix of $F_{k+1} \uplus (D \setminus S)$ will be strictly more than r by Observations 5 and 6, contradicting the fact that S is a solution set. Finally, observe that if $D \setminus S$ consists of isolated vertices, then S is a vertex cover of G. This completes the proof. □

3.4.2 An FPT algorithm and kernel for r-Rank Vertex Deletion

We start by recalling that the rank of any skew-symmetric matrix is always even. In the rest of the chapter we always assume that target rank r is even. We will be using the following result that characterizes skew-symmetric matrices that have rank less than or equal to r.

Theorem 16 (see Chapter 7 of [98])**.** *For an even positive integer r, a skew-symmetric*

matrix A has rank at most r if and only if all its submatrices of size $(r+2)\times(r+2)$ *have rank at most r.*

We obtain an algorithm and a kernel by reducing r-Rank Vertex Deletion to $(2r+4)$−Hitting Set, which has a parameterized algorithm and a polynomial kernel [1]. Given a set family $\mathcal{F}$ over a universe U, a set S is said to be a *hitting set* of $\mathcal{F}$ if $S \cap F \neq \emptyset$, $\forall F \in \mathcal{F}$. An input to d−Hitting Set consists of $(U, \mathcal{F}, k)$ where $\mathcal{F}$ is a family of subsets of U of size at most d and the objective is to check if there exists a set $S \subseteq U$ of size at most k which is a hitting set of $\mathcal{F}$. Given an oriented graph D with adjacency matrix A_D, define the set family $\mathcal{H}_r(D)$ as follows,

$$\mathcal{H}_r(D) \triangleq \{X \cup Y : X, Y \subseteq V(D), |Y| = |X| = \mathsf{rank}(A_D[X,Y]) = r+2\} \tag{3.3}$$

Lemma 19. *An input instance* (D,k) *is a yes instance of* r-Rank Vertex Deletion *if and only if* $(\mathcal{H}_r(D),k)$ *is a yes instance of* $(2r+4)$-Hitting Set.

Proof. In the forward direction, suppose S is a solution for an instance (D,k) of r-Rank Vertex Deletion. We claim that S is a *hitting set* of $\mathcal{H}_r(D)$. Suppose not, then there exists a set $H \in \mathcal{H}_r(D)$ which is disjoint from S. The definition of $\mathcal{H}_r(D)$ shows that H corresponds to a submatrix of A_D which is a submatrix of $A_{D\setminus S}$ as well. Therefore by Theorem 16, $\mathsf{rank}(A_{D\setminus S}) > r$, which contradicts the fact that S is a solution for (D,k).

In the backward direction, suppose S is a hitting set of $\mathcal{H}_r(D)$. If S is not a solution of (D,k) then, by Theorem 16, there exists a submatrix $A_{D\setminus S}[X,Y]$ of $A_{D\setminus S}$ of size $(r+2)\times(r+2)$ having full rank. By the definition of $\mathcal{H}_r(D)$, $X \cup Y$ must be contained in it. As $X \cup Y$ is disjoint from S, S is not a hitting set of $\mathcal{H}_r(D)$, a contradiction. □

Lemma 19 allows us to reformulate our problem as $(2r+4)$-Hitting Set and thus using the known algorithm and kernel for the problem (see Theorem 2 in [1], Chapter 2 in [27] or Theorems 8 and 9 in [80]) we get the following result.

Theorem 17. *r-Rank Vertex Deletion admits a kernel of size $O(k^{2r+4})$ and an algorithm with running time $2^{O(k \log r)} n^{O(1)}$.*

Proof. For any fixed r, the collection $\mathcal{H} = \mathcal{H}_r(D)$ can be enumerated in $n^{O(r)}$ time. The set $\mathcal{H}$ is not a uniform family. The cardinality of each set in $\mathcal{H}$ is a number between $r + 2$ and $2(r + 2)$. We make the family uniform by padding each set in $\mathcal{H}$ with distinct new elements increasing the cardinality to $2(r + 2)$. These 'new' elements occur exactly once in each set and the sets contain at least $r + 2$ 'old' elements. Any new element can hit exactly one set of the family. Suppose some solution picks a 'new' element, one can replace it by any other 'old' element in the set. The new elements are added just to make the set family uniform.

If $|\mathcal{H}| > (2r+4)!(k+1)^{2r+4}$, by the well known Sunflower Lemma [61], there is a sunflower [2] in $\mathcal{H}$ with $k + 2$ petals. Any hitting set of size k must hit each of the $k + 2$ sets present in the sunflower, therefore it must pick an element from the core of the sunflower, moreover any core will not contain any of the new elements. If the core is empty, then there is no hitting set of size k for $\mathcal{H}$ and we return no. Otherwise, delete one of the sets of the sunflower. We repeat this process until we are left with at most $(2r + 4)!(k + 1)^{2r+4}$ sets. Let $\mathcal{H}'$ be a sub-collection of $\mathcal{H}$, such that all the sets in $\mathcal{H} \setminus \mathcal{H}'$ may be safely ignored and all the new elements have been deleted. Observe that $\mathcal{H}$ has a hitting set of size k if and only if $\mathcal{H}'$ has a hitting set of size k. Since each set in $\mathcal{H}'$ contains at most $2(r + 2)$ vertices there are at most $2(r + 2)(2r + 4)!(k + 1)^{2r+4}$ vertices corresponding to $\mathcal{H}'$. Let D' be the graph induced by the vertices corresponding to $\mathcal{H}'$ in D. Observe that $\mathcal{H}'$ is a subset of the collection $\mathcal{H}_r(D')$.

Any solution S to the instance (D', k) of r-Rank Vertex Deletion is a hitting set of of $\mathcal{H}_r(D')$ having size at most k, which is also a hitting set of $\mathcal{H}'$. Any hitting set of $\mathcal{H}'$ is also a hitting set of $\mathcal{H}$, therefore S is a solution to the instance (D, k). Conversely, since

[2] A sunflower with k petals and a core Y is a collection of sets $S_1, \ldots, S_k$ such that $S_i \cap S_j = Y, \forall i \neq j$; the sets $S_i \setminus Y$ are called petals, and we require that none of the sets S_i is empty.

D' is an induced subgraph of D, any solution to (D, k) is a solution to (D', k). We output D' as the kernel.

The running time of the algorithm follows from Theorem 8 in [80]. □

3.5 Deleting arcs to reduce rank

This section considers the problem of rank reduction of the adjacency matrix of an oriented graph D by deletion of arcs. We prove several properties for the general case of editing arcs. In the case of editing a graph, a solution F is a subset of $V(D) \times V(D)$ and if an arc of F is not in D, then it is added to D, otherwise it is deleted from D. As D is an oriented graph, only one of the arcs (u, v) or (v, u) can be present in F. The graph D edited using F is denoted using $D \triangle F$. Note that deletion corresponds to the case when $F \subseteq E(D)$. Let the modules of $\sim$ in D be $V_1, \ldots, V_t$.

Definition 5 (Minimal Edit)**.** *We call F a* minimal *edit if $D \triangle F$ is an oriented graph and for all $i \in [t]$ it*

- *either contains all the arcs in $V_i \times V_j$ or none of them, for all $j \in [t]$, and*
- *does not contain the arc (u, v), for all $u, v \in V_i$.*

We call a vertex $v \in V(D)$ an *affected* vertex if it receives an edit i.e. there exists an $e \in F$ such that v is contained in the arc e. Let F be a set of edits on D and $u \in V(D)$ be any vertex, we use

$$J_{in}(u) = \{w : w \in V(D), (w, u) \in F\} \text{ and}$$

$$J_{out}(u) = \{w : w \in V(D), (u, w) \in F\}$$

to denote the set of affected vertices due to edits on arcs incident on u.

Lemma 20. *Let the equivalence classes of $\sim$ in D be $V_1, \ldots, V_t$, and the equivalence classes of $\sim$ in $D' = D \triangle F$ be $U_1, \ldots, U_s$, for some set $F \subseteq V(D) \times V(D)$. Then, F is a* minimal *edit of D if and only if for all $i \in [t]$ there exists a unique $j \in [s]$ such that $V_i \subseteq U_j$.*

Proof. Let $w \in V(D)$ be any vertex and let $i \in [t]$. Since F is a *minimal* edit, for all $v \in V_i$, exactly one of the following happens; either $(v, w) \in F$, $(w, v) \in F$ or v is not affected by any edit performed on w. This implies that every vertex in V_i has the same neighborhood in D', therefore V_i is contained in some equivalence class of D'. This proves the forward direction.

For the backward direction, assume that F is not a *minimal* edit. Then, there exist $i, j \in [t]$ with $i \neq j$ and $x \in \{in, out\}$, such that there is a vertex $w \in V_j$ for some $u, v \in V_i$ with $w \in J_x(u) \setminus J_x(v)$. In words, this means that u and v have different neighborhoods which implies that u and v are in different equivalence classes in D', which is a contradiction. □

Lemma 21. *Let D be an oriented graph with $V_1, \ldots, V_t$ as modules of $\sim$ in D. For any vertex $v \in V_i$, let $D^v_{i,j}$ be the graph obtained by moving vertex v from module V_i to any other module V_j and performing necessary edits to make its neighborhood same as any other vertex in V_j. Then* $\mathsf{rank}(A_{D^v_{i,j}}) \leq \mathsf{rank}(A_D)$.

Proof. The operation of moving a vertex consists of three operations, deleting v from D, copying a vertex $u \in V_j$ and relabeling the copy as v. Relabeling a vertex does not change the rank and the result follows by Lemma 17 and Corollary 7. □

Lemma 22. *Any set of edits F on an oriented graph D can be transformed into a* minimal *set of edits F' of D such that $|F'| \leq |F|$ and* $\mathsf{rank}(A_{D \triangle F'}) \leq \mathsf{rank}(A_{D \triangle F})$.

Proof. Let the modules of $\sim$ in D be $V_1, \ldots, V_t$. We execute the following procedure on D and F until we obtain a *minimal* edit.

1. Initialize $a = 1$.

2. Let $F_a = F$ and $D_a = D \triangle F_a$.

3. If F_a is not a minimal edit, then repeat the following.

   - Let the modules of $\sim$ in D_a be $U_1, \ldots, U_s$.
   - By Lemma 20, there exists a V_i such that $V_i \cap U_h \neq \emptyset$ and $V_i \cap U_j \neq \emptyset$ for some $h \neq j$, $h, j \in [s]$.
   - For $x \in \{in, out\}$, let $J_x^a(u)$ denote the edits received by u due to the edits in F_a. Let $C_a(u) = |J_{in}^a(u) \cup J_{out}^a(u)|$ be the number of edits performed on a vertex u by F_a.
   - Let $v_a = \text{argmin}_{y \in V_i} C_a(y)$ be a vertex receiving the minimum number of edits amongst all the vertices in V_i.
   - Assume w.l.o.g. that $v_a \in U_h$.
   - Move every vertex $w \in V_i \setminus U_h$ to the module U_h. Denote the new graph by D_{a+1}.
   - Let F_{a+1} be the new set of edits required to get D_{a+1} from D. By Lemma 21, $\mathsf{rank}(A_{D_{a+1}}) \leq \mathsf{rank}(A_{D_a})$.
   - Increment the value of a by 1.

Notice that the rank of graph D_a never increase inductively. The procedure stops when we have obtained a *minimal* edit and the procedure above terminates in a finite number of steps as it decreases the number of modules which violate the minimality condition in each iteration. We now prove that for each a the number of edits $|F_{a+1}| \leq |F_a|$. For each V_i, as v_a is the vertex with minimum number of edits in it, moving any $u \in V_i \setminus U_h$ to U_h does not increase the number of edits. Each vertex u receives the same number of edits

as the number of edits received by v_a in the graph D_{a+1}. Notice that the number of edits received by v_a in F_{a+1} is $C_{a+1}(v_a) \leq C_a(v_a)$, as any arcs between u and v_a get deleted in D_{a+1}. □

Using Lemma 22 we get the following.

Corollary 9. *Any solution F of an instance (D, k) of r-*Rank Edge Deletion *is disjoint from the arcs incident on any module having strictly more than k vertices.*

3.5.1 NP-Completeness

In this section we prove the hardness result for the arc deletion problem r-Rank Edge Deletion; we show that it is NP-Complete for $r \geq 2$. We will first prove a lemma which characterises bipartite oriented graphs of rank 2.

Definition 6. *A* bi-digraph *is an oriented graph D whose vertex set consists of two disjoint sets V_1 and V_2 and its arc set $E(D)$ is a subset of $(V_1 \times V_2) \cup (V_2 \times V_1)$. A* simple bi-digraph *is a* bi-digraph *having at least one edge, such that for $i \in [2]$ and any two non-isolated vertices $a, b \in V_i$, exactly one of the following conditions holds,*

i) $N^-(a) = N^-(b)$ *and* $N^+(a) = N^+(b)$.

ii) $N^-(a) = N^+(b)$ *and* $N^+(a) = N^-(b)$.

Note that a directed bi-clique is a bi-digraph as well as a simple bi-digraph.

Lemma 23. *A bi-digraph is a simple bi-digraph if and only if the rank of its adjacency matrix is* 2*. In addition, a simple bi-digraph is the disjoint union of isolated vertices and a non-empty oriented graph D such that the underlying undirected graph of D is a complete bipartite graph.*

Proof. A bi-digraph D, by construction, has an adjacency matrix of the following form

$$A_D = \begin{bmatrix} \mathbf{0} & A \\ -A^T & \mathbf{0} \end{bmatrix}, \tag{3.4}$$

where A is a matrix whose rows are indexed by V_1 and the columns are indexed by V_2 and $\mathbf{0}$ is the all zeroes matrix of appropriate size.

For the forward implication, observe that $\mathsf{rank}(A_D) = 2 \cdot \mathsf{rank}(A)$. Therefore it suffices to prove that $\mathsf{rank}(A) = 1$. By Definition 6, for any two non-isolated vertices $a, b \in V_2$, the columns A^a and A^b are related as $A^a = \pm A^b$. Therefore, a column of A is either all-zero or one of $\pm A^a$, which implies that $\mathsf{rank}(A) = 1$.

For the backward implication, suppose D is a bi-digraph with $\mathsf{rank}(A_D) = 2$, which implies that $\mathsf{rank}(A) = 1$. Therefore, there exists a non-zero column A^j such that every column of A is contained in its span. As the entries of A are restricted to take values from $\{-1, 0, 1\}$, each column of A is equal to one of A^j, $-A^j$ or the all-zero column. Thus D is a simple bi-digraph.

To prove the last property, observe that deleting the all-zero columns and rows (which correspond to isolated vertices) from A results in a matrix which consists exclusively of $+1$ or -1. The adjacency matrix of the underlying undirected graph is obtained by mapping each -1 to $+1$ in the adjacency matrix of an oriented graph, in this case it results in the adjacency matrix of an undirected complete bipartite graph. □

The problem r-Rank Edge Deletion is polynomial time solvable for $r = 0$ as the solution consists of deleting all the arcs. We prove that r-Rank Edge Deletion is NP-Complete for any $r \geq 2$.

Theorem 18. *r-Rank Edge Deletion is NP-Complete for $r \geq 2$.*

Proof. We give a reduction from the Maximum Bi-Clique problem which was shown to

be NP-Complete by Peeters [87]. The Maximum Bi-Clique problem takes a loopless undirected graph G and a positive integer K as input and outputs "yes" if and only if there exists a bi-clique (a complete bipartite graph) on disjoint vertex sets $A, B \subseteq V(G)$ such that $|A| \times |B| \geq K$.

Given an undirected graph G construct the bi-digraph D as follows: the vertex set $V(D) = \{u_i : u \in V(G), i \in [2]\}$ and arc set $E(D) = \{(u_1, v_2) : (u, v) \in E(G)\}$. For $i \in [2]$, let $V_i = \{x_i : x \in V(G)\}$. For any $x_i \in V(D)$ for $i \in [2]$, we use the unsubscripted symbol x to denote the corresponding vertex in the undirected graph G. Denote the number of arcs in D by $m = 2 \cdot |E(G)|$. Let (G, K) be an instance of the Maximum Bi-Clique problem then $(D, m - K)$ is the input instance of 2-Rank Edge Deletion.

For the forward implication, assume (G, K) has a bi-clique on the sets $A, B \subset V(G)$. Let $S = \{(a_1, b_2) : a \in A, b \in B\}$ be a set of arcs; since (A, B) is a solution, $|S| \geq K$. Now $S' = E(D) \setminus S$ is a solution of $(D, m - K)$ as $D \setminus S'$ is a directed bi-clique from $A_1 = \{a_1 : a \in A\}$ to $B_2 = \{b_2 : b \in B\}$ and is also a simple bi-digraph, by Lemma 23 rank of its adjacency matrix is 2.

For the backward implication, let S be a solution for $(D, m - K)$. By construction, the remaining number of arcs in $D \setminus S$ is more than K. Since $\mathsf{rank}(A_{D \setminus S}) = 2$ and $D \setminus S$ is a bi-digraph, by Lemma 23 it is a simple bi-digraph and consists of disjoint union of isolated vertices and a directed bi-clique over sets $A_1 \subseteq V_1$ and $B_2 \subseteq V_2$. Let $A = \{a : a_1 \in A_1\}$ and $B = \{b : b_2 \in B_2\}$ be subsets of $V(G)$. We claim that (A, B) is a solution of (G, K). Observe that $A \cap B = \emptyset$, otherwise let $x \in A \cap B$. As A_1 and B_2 induce a directed bi-clique in D, (x_1, x_2) is an arc in D which is not possible as that implies (x, x) is an edge in G which contradicts the assumption that G is a loopless undirected graph. Therefore, (A, B) is a solution of (G, K).

The statements above prove that r-Rank Edge Deletion is NP-Complete for $r = 2$. To prove it NP-Complete for arbitrary $r \geq 4$ we employ a trick similar to Theorem 15. Let F be a reduced oriented graph with $\mathsf{rank}(A_F) = r - 2$. For an instance (G, K) of the Maximum

Bi-Clique problem, obtain the graph F_{m-K+1} by making $m - K + 1$ copies, keeping the neighborhood same, of each vertex in F, observe that $\mathsf{rank}(A_{F_{m-K+1}}) = r-2$ by Corollary 7. Let $(D \uplus F_{m-K+1}, m - K)$ be the input instance of r-Rank Edge Deletion. By Corollary 9, any solution of this instance must not affect any edge of F_{m-K+1}. Therefore, all the edits must be performed on D and since the target rank is r, by Observation 5, the graph D after deletion of edges must result in a graph having rank at most 2. Using these arguments, the equivalence of instances follows in an easy manner. □

3.5.2 A parameterized algorithm for r-Rank Edge Deletion

In this section we design a polynomial kernel and a sub-exponential parameterized algorithm for r-Rank Edge Deletion.

Lemma 24. *For any instance (D, k) of the r-Rank Edge Deletion problem, if $D^{\sim}$ has more than $3^r + 2k$ vertices then (D, k) is a no instance.*

Proof. Let (D, k) be a yes instance and D' be the graph obtained from D after deleting a *minimal* solution having at most k arcs. The graph D' has rank r, by Corollary 6 it has at most 3^r modules. Any *minimal* solution having size at most k can affect at most $2k$ modules of D. If two distinct modules of D end up in the same module of D' then at least one of them must be affected. Every module of D ends up in some module of D', therefore there can be at most $3^r + 2k$ modules in D. □

An application of Lemma 24 and Corollary 9 results in the following kernel for r-Rank Edge Deletion.

Theorem 19. *r-Rank Edge Deletion admits a kernel with $\tilde{O}((3^r + 2k) \cdot (k + 1))$ vertices.*

Proof. Let (D, k) be an input instance of r-Rank Edge Deletion. If D has more than $3^r + 2k$ modules then by Lemma 24 it is a no instance. Otherwise, construct the graph D'

by removing all but $k+1$ vertices from each module of D, we call the modules whose vertices were deleted as *modified* modules. We will show that the instances (D, k) and (D', k) are equivalent.

In the forward direction, observe that the graph D' is a subgraph of D hence any solution of (D, k) is also a solution of (D', k). Next, we prove the backward direction of the equivalence of instances. Let F' be a minimal solution of (D', k). By Corollary 9, F' does not affect any vertex in any of the modified modules of D'. Therefore, the reduced graphs of $D' \setminus F$ and $D \setminus F$ are isomorphic and have the same rank. This implies that F' is also a solution of (D, k).

We output (D', k) as the desired kernel. The instance (D', k) has at most $3^r + 2k$ equivalence classes and each module has at most $k+1$ vertices, so the claimed bound on the size $V(D')$ follows. □

The family of undirected $\sim$-reduced graphs of rank equal to r can be constructed in time which is a function of r alone [2]. With very minor changes in the algorithm (Section 3, [2]), we have the following theorem.

Theorem 20. *The family $\mathcal{D}_r$ of reduced oriented graphs of rank at most r can be constructed in time which is a function of r alone.*

Oriented H-Bag Deletion To solve our problem we will be making use of an algorithm for the Oriented H-Bag Deletion problem which is defined as follows.

Oriented H-Bag Deletion **Parameter:** k
Input: Oriented graphs D, H and an integer k, where H is a $\sim$-reduced graph.
Question: Does there exist a set F with at most k arcs of D such that the $\sim$-reduced graph of $D' = (V(D), E(D) \setminus F)$ is an induced subgraph of H ?

The rest of the section gives an algorithm for Oriented H-Bag Deletion and shows how to use it for solving r-Rank Edge Deletion. Our algorithm for Oriented H-Bag Deletion

is similar to the bag-editing algorithm of Damaschke et. al. [29] but is novel in the sense that it handles a different equivalence relation and works with oriented graphs. We now present the algorithm for Oriented H-Bag Deletion.

We begin with a known lemma about branching vectors and then proceed to the algorithm.

Lemma 25 (Lemma 1, [29])**.** *The branching vector* $(1, r, \ldots, r)$*, with* r *repeated* q *times, has a branching number bounded by* $1 + \frac{\log_2 r}{r}$ *if* r *is large enough compared to the fixed* q*.*

Theorem 21. *The* Oriented H-Bag Deletion *problem with a fixed graph* H *and an input instance* (D, k) *can be solved in* $2^{O(\sqrt{k}\log k)}n^{O(1)}$ *time.*

Proof. To each vertex of H we associate a bag which will be filled in with the vertices of D while respecting the adjacency relations of H. Let $b = |V(H)|$ denote the number of bags in H and use $\mathcal{B} = \{B_i : i \in [b]\}$ to denote the set of bags. Assume that the vertices in H are $v_1, \ldots, v_b$ and B_i corresponds to the vertex v_i, for all $i \in [b]$. The set of $\sim$-modules of D is denoted using $\mathcal{M} = \{V_1, \ldots, V_t\}$, where $t \leq b+2k$ by Lemma 24. In the preprocessing phase of the algorithm we first add vertices to bags and then add arcs which are a subset of arcs in D to simulate deletion of arcs.

Preprocessing Phase: The bags can be in two states "closed" or "open". If a bag is marked *open*, then we are allowed to add vertices in it, otherwise is is marked *closed*. At the start of the algorithm all the bags are empty and open. In every bag we create $s = \lfloor \sqrt{k} \rfloor$ empty slots to be filled later, thus, there are a total of $b \cdot s$ slots to be filled. We call a bag *free* if it is open and less than s slots have been filled in it. The state of a node in the branching tree is denoted using $(\mathcal{M}, \mathcal{B})$. In the i^{th} branch, for $i \in [t]$, pick an arbitrary vertex u from V_i, delete it from V_i and add it to the lowest number free bag in $\mathcal{B}$. In the $(t+1)^{th}$ branch we mark the lowest number free bag as closed. We stop the branching when no bag is free. At each leaf node $(\mathcal{M}, \mathcal{B})$ of the branching tree, we construct a graph H' over the vertices in $\cup\mathcal{B}$. For every arc (v_i, v_j) in H we add all the

arcs $B_i \times B_j$ in the graph H'. Next we check if the neighborhood of every vertex in the graph H' is a subset of its neighborhood in $D[\cup\mathcal{B}]$, in this case we reduce the parameter k by $|E(D[\cup\mathcal{B}])| - |E(H')|$; otherwise we discard this branch and return, as this bag filling cannot be obtained by deletion of some arcs in D. This completes the description of this phase of the algorithm. Observing that there are a total of $t + 1$ branches and the depth of the branching tree is $b \times s$, we get the following.

Lemma 26. *The total running time of the preprocessing phase is bounded by* $(b + 2k + 1)^{b \cdot s} = 2^{O(\sqrt{k} \log k)}$.

Observation 7. *At the end of the preprocessing phase, if each open bag is not full then we have assigned all the vertices of D in the bags of H and we answer yes if* $k \geq 0$ *for that branch, otherwise we discard that branch. Thus, we can assume that all the open bags have exactly s vertices in them.*

Assigning bags to undecided vertices when all the bags are open: We first consider the special case when all the bags are open, later will show how to handle the general case when closed bags are also present. This phase of the algorithm begins at the leaf node $(\mathcal{M}, \mathcal{B})$ along with the graph H' and the reduced parameter k as provided by the preprocessing phase. The vertices in $V(D) \setminus \cup\mathcal{B}$ are not yet added to any bag and are called as *undecided vertices*. So far the arc deletions have been done within the bag vertices only, these arcs will remain fixed for the rest of the algorithm and we will not delete them. At the end of the algorithm we will add undecided vertices to H' along with some of the arcs present in D. In the rest of the proof we will be working with the graph H' and will be deleting edges in order to accommodate undecided vertices in the bags of H'.

We first correct the adjacency relation of every vertex u with respect to the bags. If there exists an undecided vertex u and a bag $B_i \in \mathcal{B}$ such that u and B_i does not induce a directed bi-clique then we delete all arcs between u and B_i and decrease the parameter k accordingly. We do this for all such undecided vertices. Note that we do this without

branching. For now, u is not yet added to any bag. Now every undecided vertex u is either completely adjacent or completely non-adjacent to each bag in $\mathcal{B}$.

Fitting undecided vertices in bags: We say that u *fits* in a bag $B(u) \in \mathcal{B}$, if u has the same neighborhood as vertices in $B(u)$ in the graph induced on bags. As H is a reduced graph and all the vertices have distinct neighborhoods, every vertex u fits in at most one bag. If there exists an undecided vertex u that fits in no bag, we branch and decide on a bag for u, put u in this bag, and do the necessary deletions. If deletions cannot put u in this bag, we discard this branch. Since u does not fit in any bag, we need at least s deletions as each open bag has s vertices by Observation 7. Thus the branching vector, of length b, is $(s, \ldots, s)$ or better. Now, every undecided vertex u fits in exactly one bag $B(u)$. Remember that we can not add undecided vertices to bags yet, as we may need to correct the adjacency relation amongst the undecided vertices; this is done in the next paragraph.

Correcting adjacency amongst the undecided vertices: Suppose that two undecided vertices u and v have the "wrong" adjacency relation. We now enumerate all the possibilities and show how to handle them.

- If (u, v) is an arc but $B(u)$ and $B(v)$ are not adjacent, then we delete the arc (u, v) and decrease k by 1.
- There are two more cases to consider,
  - If (u, v) is not an arc but $B(u)$ and $B(v)$ are adjacent, or
  - If (v, u) is an arc but $B(u)$ has all the arcs to $B(v)$.

  Since u and v cannot be simultaneously put in their respective bags, we decide on a new bag for both u and v and branch if both of them can fit in with deletion of arcs in the new bag. For the second case above, we have the option of deleting the arc as well, we consider both branches i.e. in one branch we delete the arc and in second branch we keep it. We keep on repeating these steps in order to correct the

adjacency relation amongst undecided vertices until we exhaust the deletion budget k.

Putting undecided vertices in bags: If $k < 0$ in any branch above, then discard it. If there is a branch such that all the vertices have the correct adjacency relation pairwise and $k \geq 0$, then we can put the undecided vertices in the bags they fit in and answer "yes" for the instance. We answer "no" if none of the branches output "yes". Note that we also keep track of deleted arcs. The branching vector is $(1, s, \ldots, s)$, with s repeated at most $2b$ times, or better.

Lemma 27. *For the branches which have all the bags open at the end of the preprocessing phase, the branching vector is* $(1, s, \ldots, s)$*, with* s *repeated at most* $2b$ *times, or better.*

Filling bags when closed bags are present: The closed bags do not guarantee at least s deletions when we are fixing bags for undecided vertices. This paragraph will show that they can be handled after the open bags have been dealt with.

Dealing with open bags: Let U be the set of vertices of H' corresponding to the open bags. In this case, $H'[U]$ may not remain a $\sim$-reduced graph. In that case we group together bags which are $\sim$-equivalent in $H'[U]$ and call each group a *superbag*. So each bag in $H'[U]$ is either a "normal" bag or a superbag. Observe that each superbag has at least s vertices in it. On $H'[U]$ containing the superbags, we perform exactly the same branching rules as described for the case when all bags are open. We have fewer branches and the branching vectors do not get worse. After branching, we have that the undecided vertices fit in one of the bags in $H'[U]$ and they have correct adjacency relation amongst themselves.

Dealing with closed bags: While actually adding a vertex u to a bag we also decide on the bag within a superbag S that will host u. Adding u to a bag does not change the adjacency relations within the union of open bags and undecided vertices. Therefore we can take these decisions independently for all u, and add it to any bag in S that causes the

minimum number of arc deletions between u and the closed bags. We discard the branch if u cannot be added to a bag in S with deletion of arcs between it and the closed bags. The worst branching vector we encounter anywhere in the algorithm is $(1, s, \ldots, s)$ with s repeated at most $2b$ times. Combining this with Lemmata 25 and 27, we obtain the bound on the running time for assigning undecided vertices to be $(1 + \frac{\log \sqrt{k}}{\sqrt{k}})^k = 2^{O(\sqrt{k} \log k)}$. Multiplying it with the running time of preprocessing phase, as given by Lemma 26, we get the claimed running time.

Finally, observe that the algorithm terminates as each branch decreases k by at least 1. It is also straighforward to check that if (D, k) has a solution S then the algorithm outputs either an equal sized or a smaller solution. □

Making use of the algorithm above we get the algorithm for r-Rank Edge Deletion.

Theorem 22. *An instance* (D, k) *of* r-Rank Edge Deletion *can be solved in* $2^{O(\sqrt{k} \log k)} n^{O(1)}$ *time.*

Proof. We generate family $\mathcal{D}_r$ of $\sim$-reduced graphs using Theorem 20. For each graph $H \in \mathcal{D}_r$, we check if arc deletions can convert D into a graph having H as reduced graph using Theorem 21. If the answer is "no" for all H, we output "no", otherwise we output "yes". For the correctness of algorithm observe that after k deletions the reduced graph of D is a member of $\mathcal{D}_r$ if and only if it's rank is at most r. □

Chapter 4

Matrix Rigidity from the Viewpoint of Parameterized Complexity[1]

The *rigidity* of a matrix A over a field $\mathbb{F}$ is the minimum Hamming distance between A and a matrix of rank at most r. Rigidity is a classical concept in Computational Complexity Theory: constructions of rigid matrices are known to imply lower bounds of significant importance relating to arithmetic circuits. Yet, from the viewpoint of Parameterized Complexity, the study of central properties of matrices in general, and of the rigidity of a matrix in particular, has been neglected. In this chapter, we conduct a comprehensive study of different aspects of the computation of the rigidity of *general matrices* in the framework of Parameterized Complexity. Naturally, given a parameter k, the Matrix Rigidity problem asks whether the rigidity of A is a most r. In case $\mathbb{F} = \mathbb{Q}$ or the edited entries must contain integers, to the best of our knowledge, it is not even known whether the problem is decidable [95]. Surprisingly, we show that in case $\mathbb{F} = \mathbb{R}$ or $\mathbb{F}$ is any finite field, this fundamental problem is fixed-parameter tractable with respect to $k + r$. To this end, we present a simple yet powerful dimension reduction procedure, which we believe to be a valuable primitive in future studies of problems of this nature. We further employ central

[1]This chapter is based on a joint work with Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh and Meirav Zehavi [40].

tools in Real Algebraic Geometry, which are not well known in Parameterized Complexity, thus establishing their ability to form a bridge between Matrix Theory and Parameterized Complexity. In particular, we view the output of our dimension reduction procedure as an algebraic variety. Our main results are complemented by a W[1]-hardness result and a subexponential-time parameterized algorithm for a special case of Matrix Rigidity, highlighting the different flavors of this problem.

4.1 Introduction

The *rigidity of a matrix* is a classical concept in Computational Complexity Theory, which was introduced by Grigoriev [49, 50] in 1976 and by Valiant [99] in 1977. Constructions of rigid matrices are known, for instance, to imply lower bounds of significant importance relating to arithmetic circuits. Yet, from the viewpoint of Parameterized Complexity, the study of central properties of matrices in general, and of the rigidity of a matrix in particular, has been neglected. The few papers that do consider such properties are restricted to the very special case of adjacency matrices, and therefore they are primarily studies in Graph Theory rather than Matrix Theory [80, 81]. In this chapter, we conduct a comprehensive study of different aspects of the computation of the rigidity of *general matrices* in the framework of Parameterized Complexity.

Formally, given a matrix A over a field $\mathbb{F}$, the rigidity of A, denoted by $\mathcal{R}_A^{\mathbb{F}}(r)$, is defined as the minimum Hamming distance between A and a matrix of rank at most r. In other words, $\mathcal{R}_A^{\mathbb{F}}(r)$ is the minimum number of entries in A that need to be edited in order to obtain a matrix of rank at most r. Naturally, given parameters r and k, the Matrix Rigidity problem asks whether $\mathcal{R}_A^{\mathbb{F}}(r) \leq k$. In the case when $\mathbb{F} = \mathbb{Q}$ or the edited entries must contain integers, it is not even known whether the problem is decidable [95]. We therefore focus on the cases where $\mathbb{F} = \mathbb{R}$ or $\mathbb{F} = \mathbb{F}_p$ is a finite field for some prime p. Formally, we study the following forms of Matrix Rigidity. Here, FF Matrix Rigidity is not restricted to a

specific finite field $\mathbb{F}_p$, but includes $\mathbb{F}_p$ as part of the input.

Real Matrix Rigidity	**Parameter:** r, k
Input: A matrix A with each entry an integer, and two non-negative integers r, k.	
Question: Is $\mathcal{R}_A^{\mathbb{R}}(r) \leq k$?	

FF Matrix Rigidity	**Parameter:** p, r, k
Input: A finite field $\mathbb{F}_p$ of order p, a matrix A over $\mathbb{F}_p$, and two non-negative integers r, k.	
Question: Is $\mathcal{R}_A^{\mathbb{F}_p}(r) \leq k$?	

Valiant [99] presented the notion of the rigidity of a matrix as a means to prove lower bounds for linear algebraic circuits. He showed that the existence of an $n \times n$ matrix A with $\mathcal{R}_A^{\mathbb{F}}(\epsilon n) \geq n^{1+\delta}$ would imply that the linear transformation defined by A cannot be computed by any arithmetic circuit having size $O(n)$ and depth $O(\log n)$ in which each gate of the circuit computes a linear combination of its inputs. Later, Razborov [90] (see [72]) established relations between lower bounds on rigidity of matrices over the reals or finite fields and strong separation results in Communication Complexity. Although many efforts have been made in this direction [43, 97, 73, 67] (this is not an exhaustive list), proofs of good lower bounds for explicit families of matrices still remain elusive. For a recent survey on this topic, we refer the reader to [74]. The formulation of Matrix Rigidity as stated in this chapter was first considered by Mahajan and Sarma [78], and it was shown to be NP-Hard for any field by Deshpande [31]. In this chapter, we study the concept of the rigidity of a matrix from a different perspective, given by the framework of Parameterized Complexity (see Section 4.2).

We remark that, in Chapters 2 and 3, we studied the following problems, which are related to Matrix Rigidity but are simpler as they are restricted to graphs. Given a graph $G = (V, E)$ and two non-negative integers r, k, the problem r-Rank Vertex Deletion (r-Rank Edge Deletion) asks whether one can delete at most k vertices (resp. edges) from G so

that the rank of its adjacency matrix would be at most r, while r-Rank Edge Editing asks whether one can edit k edges in G so that the rank of its adjacency matrix would be at most r.[2] For undirected graphs, in earlier chapters, we also proved that these problems are NP-Hard even if r is fixed, but they can be solved in time $O^*(2^{O(k \log r)})$. We also showed that r-Rank Edge Deletion and r-Rank Edge Editing can be solved in time $O^*(2^{O(f(r)\sqrt{k}\log k)})$. We also obtained similar results for directed graphs in Chapter 3.

Our Contribution In this chapter, we establish that both Real Matrix Rigidity and FF Matrix Rigidity are FPT with respect to $r+k$. Specifically, we obtain the following results.

Theorem 23. Real Matrix Rigidity *can be solved in* $O^*(2^{O(r\cdot k\cdot \log(r\cdot k))})$ *time.*

Theorem 24. FF Matrix Rigidity *can be solved in* $O^*(f(r,k))$ *time for a function* f *that depends only on* r *and* k.

Observe that the dependency of the running times on the dimension of the input matrix is polynomial, and in the case of FF Matrix Rigidity, the dependency of the running time on p is also polynomial. In the case of Real Matrix Rigidity, the dependency of the running time on the maximum length (in binary) of any entry in *both* input and output matrices is polynomial. In this context, recall that in case $\mathbb{F} = \mathbb{Q}$ or the edited entries must contain integers, it is not even known whether Matrix Rigidity is decidable [95]. We also show that,

Theorem 25. FF Matrix Rigidity *is solvable in time* $O^*(2^{O(f(r,p)\sqrt{k}\log k)})$ *for some function* f *that depends only on* r *and* p.

Here, the dependency of the running time on k is subexponential, but the dependency of the running time on p is unsatisfactory in case p is not fixed. This algorithm adapts ideas from Chapters 2 and 3.

[2]Editing an edge $\{u, v\}$ means that if $\{u, v\} \in E$ then $\{u, v\}$ is deleted, and otherwise it is added.

To obtain our main results, we first present a dimension reduction procedure, which we believe to be a valuable primitive in future studies of problems of this nature. Our procedure is simple to describe and given an instance of Matrix Rigidity, it outputs (in polynomial time) an equivalent instance where the matrix contains at most $O((r \cdot k)^2)$ entries. Furthermore, the set of entries of the output matrix is a subset of the set of entries of the input matrix. We believe this procedure to be of interest independent of our main results as it establishes that FF Matrix Rigidity admits a polynomial kernel with respect to $r+k+p$. The simplicity of our procedure also stems from its modularity—it handles rows and columns in separate phases. On a high-level, this procedure is defined as follows. For $k+1$ steps, it repeatedly selects a set of maximum size consisting of rows that are linearly independent, where if the size of this set exceeds $r+1$, it is replaced by a subset of size exactly $r+1$. Each such set of rows is removed from the input matrix, and then it is inserted into the output matrix. At the end of this greedy process, rows that remain in the input matrix are simply discarded. The correctness of our procedure relies on two key insights: **(i)** if the input instance contains more than $k+1$ pairwise-disjoint sets of rows that are linearly independent, and each of these sets is of size at least $r+1$, then the input instance is a No-instance; **(ii)** by the pigeonhole principle, any row discarded from the input matrix belongs to the span of at least one set of rows that cannot be edited. Having an intermediate matrix with a small number of rows, the procedure applies the exact same process to the input that is the transpose of this intermediate matrix, thus overall obtaining a matrix with a small number of entries.

Armed with our dimension reduction procedure, we tackle Real Matrix Rigidity and FF Matrix Rigidity by employing central tools in Algebraic Geometry, which are not well known in Parameterized Complexity, as a black box. For this purpose, we first recall that the rank of a matrix is at most r if and only if the determinant of all of its $(r+1) \times (r+1)$ submatrices is 0. Since at this point we can assume that we have a matrix containing only $O((r \cdot k)^2)$ entries at hand, we may "guess" *which* entries should be edited. Yet, it is not clear *how* these entries should be edited. However, with the above observation in

mind, we are able to proceed by viewing our current problem in terms of an algebraic variety (such a formulation was also used in the context of complexity analysis in [95]). In particular, this viewpoint gives rise to the applicability of firmly established tools that determine the feasibility of a system of polynomials [7, 62].

Our main results are complemented by a W[1]-hardness result and a subexponential-time parameterized algorithm for a special case of Matrix Rigidity, which overall present the different flavors of this problem and the techniques relevant to its study. We show that both Real Matrix Rigidity and FF Matrix Rigidity are W[1]-hard with respect to the parameter k. (The papers [80, 81] already imply that both of these problems are para-NP with respect to the parameter r.) Our reduction is inspired by studies in Parameterized Complexity that involve the Odd Set problem [33], and consists of four reductions, one of which builds upon the recent work of Bonnet et al. [12].

The complexity of our reduction stems from the fact that unlike previous studies of this nature, we establish the W[1]-hardness of our problem of interest over *any* finite field and over the field of reals rather than only over a specific finite field. Thus, we first need to define a special case of Odd Set, which we call Partitioned Unique Intersection, and observe that its W[1]-hardness follows from the proof of the W[1]-hardness of Odd Set that is given in [33]. The correctness of our reductions crucially relies on the implications of the properties of this special case. Our first reduction translates Partitioned Unique Intersection to a problem involving matrices rather than sets, which we call Partitioned Unit Multiplication. Then, to be able to discuss any finite field as well as the field of reals, we introduce new variants of Partitioned Unit Multiplication and the Nearest Codeword problem, called $\mathbb{F}$-Unit Multiplication and $\mathbb{F}$-Nearest Codeword, respectively. The application of our second reduction results in an instance of $\mathbb{F}$-Unit Multiplication. Then, the application of our third reduction, which builds upon [12], results in an instance of $\mathbb{F}$-Nearest Codeword. Finally, we devise a reduction whose application results in an instance of Matrix Rigidity. Here, we make explicit use of the fact that the rank of the target

matrix can be large. The overall structure of the reduction may be relevant to studies of other problems where the field is not fixed.

4.2 Preliminaries

The *Hamming distance* between two strings of equal length is the number of positions at which they differ.

Linear Algebra The symbols $\mathbb{R}$, $\mathbb{Q}$ and $\mathbb{F}_p$ are used to denote the field of real numbers, the field of rational numbers, and a finite field of order p, respectively. We also use the unsubscripted symbol $\mathbb{F}$ to denote a field, in which case its order is denoted by $|\mathbb{F}|$.

We use S_n to denote the collection of all permutation functions of n elements. We call a matrix $\widetilde{A}$ a *jumbled matrix* of A if one can perform a series of row and column exchanges on $\widetilde{A}$ to obtain the matrix A. Equivalently, for an $m \times n$ matrix A and its *jumbled matrix* $\widetilde{A}$, there exist two permutations $\sigma_r \in S_m$ and $\sigma_c \in S_n$ such that $\widetilde{A}_i^j = A_{\sigma_r(i)}^{\sigma_c(j)}$, for all $i \in [m]$ and $j \in [n]$. Similarly, we call a matrix $\widetilde{A}$ a *jumbled submatrix* of A if there exists a submatrix of A which is a *jumbled matrix* of $\widetilde{A}$. A *mixed matrix* is a matrix having either an indeterminate or a value at each entry. We will be dealing with mixed matrices where the values belong to a finite field or $\mathbb{Z}$. We use I_n to denote the *identity matrix* of size $n \times n$.

System of Polynomial Equations Let $x_1, \ldots, x_n$ be variables. Then, a *monomial* is defined as a product $\prod_{i=1}^{n} x_i^{a_i}$ for non-negative integers $a_1, \ldots, a_n$. The degree of a variable x_i in a monomial $\prod_{i=1}^{n} x_i^{a_i}$ is defined to be the number a_i, for $i \in [n]$. The degree of a *monomial* is defined as the sum of degrees of each variable occurring in it. A polynomial over a field $\mathbb{F}$ consists of a sum of monomials with coefficients from the field $\mathbb{F}$. The *total degree* of a polynomial is the degree of a monomial having maximum degree. Given a system of polynomial equations $\mathcal{P} = \{P_1 = 0, P_2 = 0, \ldots, P_m = 0\}$ over a field $\mathbb{F}$, we

say that $\mathcal{P}$ is *feasible* over $\mathbb{F}$ if there exists an assignment of values from the field $\mathbb{F}$ to the variables in $\mathcal{P}$ which satisfies every polynomial equation contained in $\mathcal{P}$.

Bounded Search Trees Informally, a *bounded search tree* or *branching* is used to represent the execution of an algorithm which solves a problem based on the solution of sub-problems. It can be represented as a tree and the algorithm can be imagined to solve the sub-problems one at a time by traversing this tree. The correctness of a branching algorithm can be justified by arguing that in the case of a YES-instance some sequence of decisions captured by the algorithm leads to a feasible solution. The running time of the algorithm is given by the size of the branching tree. For a parameterized instance, if the size of the branching tree is bounded by a function of the parameter and each step of the algorithm takes polynomial time then such a branching algorithm leads to an FPT algorithm.

One method to bound the size of a branching tree employs the notion of *branching vectors*. To each node of the tree we associate a value using a function which depends on the instance to be solved at that node. This function, usually referred to as a *measure function*, is set up in such a way that it takes a smaller value for a sub-problem. It should also satisfy the property that it is a bounded function. Now the size of the tree can be upper-bounded by looking at the drop in value of the measure function at each branch of a node. This drop is represented as a tuple of numbers and is referred to as a *branching vector*. Formally, suppose that the algorithm executes a rule which has ℓ branches (each corresponding to a recursive call), such that in the i^{th} branch, the current value of the measure decreases by b_i. Then, $(b_1, b_2, \ldots, b_\ell)$ is the branching vector of this rule. We say that α is the *branching number* of $(b_1, b_2, \ldots, b_\ell)$ if it is the (unique) positive real root of $x^{b^*} = x^{b^*-b_1} + x^{b^*-b_2} + \ldots + x^{b^*-b_\ell}$, where $b^* = \max\{b_1, b_2, \ldots, b_\ell\}$. If $r > 0$ is the initial value of the measure, and the algorithm returns a result when (or before) it becomes negative, the running time is bounded by $O^*(\alpha^r)$. For more details we refer the reader to [27].

4.3 Dimension Reduction Procedure

In this section, we show how to compress an input instance of Matrix Rigidity to an equivalent instance in which the matrix has at most $O(r^2 \cdot k^2)$ entries. This is a crucial step in obtaining our FPT algorithms for Real Matrix Rigidity and FF Matrix Rigidity. In particular, this step will imply that FF Matrix Rigidity admits a polynomial kernel with respect to $r + k + p$.

Our algorithm is based on the following intuition. Suppose that A is a matrix of rank ℓ. If we could obtain a sequence $B_1, \ldots, B_k$ of pairwise disjoint sets of columns of A where each set forms a column basis of A, then the answer to the question "can we reduce the rank of A to a number $r < \ell$ by editing at most k entries in A" would have been completely determined by the answer to the same question where the editing operations are restricted to the submatrix of A formed by columns in the sets $B_1, \ldots, B_k$. The same conclusion is also true in the case where each B_i is not necessarily a basis, but simply a set of $r + 1$ linearly independent columns. Keeping this intuition in mind, we turn to examine an approach where we greedily select and remove (one-by-one) $k + 1$ pairwise disjoint sets of linearly independent columns. In each iteration, we attempt to select a set whose size is exactly $r + 1$, where if it is not possible, we select a set of maximum size.

Now, let us move to the formal part of our arguments. Note that the relation "is a jumbled matrix of" as defined in Section 4.2 is an equivalence relation. We need the following simple observations which follow from the definition of the rank of a matrix.

Observation 1. *Let $A \in \mathbb{F}^{m \times n}$ be a matrix of rank equal to r. To make the rank of A at most $r - 1$, one needs to change at least one entry in A.*

Observation 2. *For a matrix A, let $\widetilde{A}$ be a* jumbled matrix *of A. Then, the instances (A, r, k), (A^T, r, k), $(\widetilde{A}, r, k)$ and $(\widetilde{A}^T, r, k)$ are equivalent instances of* Matrix Rigidity.

Observation 3. *For a matrix A, let $\widetilde{A}$ be a* jumbled (sub-)matrix *of A. Then* $\mathsf{rank}(\widetilde{A}) = \mathsf{rank}(A)$. *If $\widetilde{A}$ be a* jumbled submatrix *of A, then* $\mathsf{rank}(\widetilde{A}) \leq \mathsf{rank}(A)$.

Using Observation 3, we have the following.

Observation 4. *If* $\widetilde{A}$ *is a* jumbled submatrix *of* A *and* $(\widetilde{A}, r, k)$ *is a* No-*instance of* Matrix Rigidity*, then* (A, r, k) *is also a* No-*instance of* Matrix Rigidity.

A solution S to an instance (A, r, k) of Matrix Rigidity is a set of size at most k consisting of tuples having three values. For an element $(i, j, e) \in S$ the value of A_i^j is set to the value e in the edited matrix. We denote the matrix edited using the solution S by A_S.

Lemma 28. *Let* $\widetilde{A}$ *be a* jumbled matrix *of an* $m \times n$ *matrix* A*. Let* $\sigma_r \in S_m$ *and* $\sigma_c \in S_n$ *be the permutations which generate the* jumbled matrix $\widetilde{A}$*. If* S *is a solution of the instance* (A, r, k) *of* Matrix Rigidity *then a solution of* $(\widetilde{A}, r, k)$ *is given by* $\widetilde{S} = \{(\sigma_r(i), \sigma_c(j), e) : (i, j, e) \in S\}$.

Proof. Using the definition of jumbled matrices and the set $\widetilde{S}$, we get that $\widetilde{A_{\widetilde{S}}}$ is a jumbled matrix of A_S. By Observation 3, we get that the $\mathsf{rank}(\widetilde{A_{\widetilde{S}}}) \leq r$. The size of $\widetilde{S}$ is equal to k, this proves that $\widetilde{S}$ is a solution of $(\widetilde{A}, r, k)$. □

As stated before, our procedure greedily selects a set of columns of A of appropriate dimension iteratively. A detailed description of the procedure, called Column-Reduction, can be found in Figure 4.1. We will now explain the ideas necessary to understand this procedure, which is the heart of this section. The input to Column-Reduction consists of a matrix A over any field, given along with non-negative integers k and r. It outputs a matrix $\widetilde{A}$ whose number of columns is bounded by a function of k and r such that the instances (A, r, k) and $(\widetilde{A}, r, k)$ are equivalent instances of Matrix Rigidity. The computation of a column basis and linearly independent vectors are done in the field $\mathbb{F}$ over which the matrix A is provided.

The procedure employs several variables. The variable i is used as an index variable whose initial value is 0, and it is incremented by 1 at a time. The case when the value of i exceeds k we will show that we are dealing with a No-instance, otherwise the value

depends on a particular input matrix A and is at most k. The variables $M_0, M_1, \ldots$ are submatrices of the input matrix A, satisfying the property that M_i is a submatrix of M_{i-1} with $M_0 = A$. In the first loop of Column-Reduction (line 3), if the matrix M_i has rank at least $r + 1$ then the variable L_i stores a set of $r + 1$ linearly independent columns in the matrix M_i. Additionally, M_i can be obtained by appending the columns in L_i to the matrix M_{i+1}. The variable $i_{\leq r}$ is set to the value of i where the rank of M_i falls below $r + 1$—after its initialization the value of $i_{\leq r}$ is not changed. In the second half of the procedure, similar to the set of variables L_i, we define a set of variables B_i which store a column basis of the matrix M_i (line 6). Recall that in this half of the procedure $i \geq i_{\leq r}$, and therefore each matrix M_i is of rank at most r. Additionally, M_i can be obtained by appending the columns in B_i to the matrix M_{i+1}, for $i \geq i_{\leq r}$. Finally, the matrix $\mathcal{L}$ is constructed using all the columns in each matrix L_i, and the matrix $\mathcal{B}$ is constructed using all the columns in each matrix B_i for appropriate values of i. By Observation 1, we have to edit at least $i_{\leq r}$ entries of $\mathcal{L}$ to make its rank at most r.

In the procedure Column-Reduction, a Yes-instance of appropriate size can be obtained by taking the matrix $Z = [0]$ (of rank 0), which contains 0 as its only entry. Clearly, (Z, r, k) is a Yes-instance of Matrix Rigidity irrespective of the values of r and k. On the other hand, the instance (I_{r+k+1}, r, k) is a No-instance of Matrix Rigidity. Therefore, the matrix I_{r+k+1} can be used in place of a No-instance of appropriate size. We need Z and I_{r+k+1} to satisfy the constraint that a kernel is an instance of the same problem as the input instance (even though, if the output is given by either line 1 or 4, we have actually solved the input instance (A, r, k) of Matrix Rigidity in polynomial time). Using the procedure Column-Reduction, it is straightforward to reduce the number of rows as well. The details of this procedure are given in Figure 4.2.

Lemma 29. *Let A be a matrix over some field $\mathbb{F}$, and let r and k be two non-negative integers. Given an instance (A, r, k), the procedure* Matrix-Reduction *runs in time polynomial in input size and returns a matrix $\widetilde{A}$ satisfying the following properties:*

1. *$\widetilde{A}$ has $O(r^2 \cdot k^2)$ entries.*

2. *If the output is produced by lines 6c and 9 of* Column-Reduction *(when called by* Matrix-Reduction*), then $\widetilde{A}$ is a* jumbled submatrix *of A.*

3. *(A, r, k) is a* Yes*-instance of* Matrix Rigidity *if and only if $(\widetilde{A}, r, k)$ is a* Yes*-instance.*

Proof. The steps of procedure Column-Reduction are all computable in polynomial time, and therefore Matrix-Reduction runs in polynomial time. We now prove the desired properties one by one. Let the matrix $\widetilde{N}$ denote the output of Column-Reduction on the input instance (N, r, k).

Proof of 1: We first bound the size of the output of Column-Reduction. The output of this procedure can occur at lines 1, 4, 6c and 9. If the output happens at line 1, it has 1 column by construction. Similarly, if the output happens at line 4, it has $r + k + 1 \leq (r + 1) \cdot (k + 1)$ columns by construction. If the output occurs at line 6c or line 9, then the number of columns in $\widetilde{N}$ is at most $(k + 1) \cdot (r + 1)$ as it is constructed using columns of at most $i \leq k$ matrices, $L_0, \ldots, L_{i_{\leq r}-1}, B_{i_{\leq r}}, \ldots, B_i$, each having at most $r + 1$ columns.

The procedure Matrix-Reduction first obtains a matrix C_A with the aforementioned number of columns by running Column-Reduction. Then, it runs Column-Reduction again on the transpose of C_A to get its rows bounded. Thus, the dimensions of the output matrix are as claimed.

Proof of 2: The relation "is a jumbled submatrix of" is a transitive relation, therefore it suffices to show that the procedure Column-Reduction outputs a jumbled submatrix of A. If the output happens at lines 6c and 9, then the columns in the output matrix are a subset of the columns in the input matrix. Therefore, in the first line of procedure Matrix-Reduction C_A is a jumbled submatrix of A. Similarly, R_A is a jumbled submatrix of C_A^T. Finally note that for matrices X and Y, X is a jumbled submatrix of Y if and only if X^T is a jumbled submatrix of Y^T. Hence, the output matrix R_A^T is a jumbled submatrix of A.

Proof of 3: We first show that the procedure Column-Reduction produces an equivalent instance of Matrix Rigidity. In the forward direction, suppose that (N, r, k) is a Yes-instance of Matrix Rigidity. If the output occurs at line 1, it is a Yes-instance by construction. The output cannot occur at line 4 as (N, r, k) is a Yes-instance. At lines 6c and 9, by property 2, Column-Reduction outputs a jumbled submatrix $\widetilde{N}$ of the input matrix N. Let S denote a solution of the instance (N, r, k) of Matrix Rigidity. By the definition of a jumbled submatrix, there exists a jumbled matrix N' of N such that $\widetilde{N}$ is a submatrix of N'. Construct a solution S' of N' from S using Lemma 28. Now construct a set $\widetilde{S}$ from S' by discarding the elements of S' with indices not occurring in the submatrix $\widetilde{N}$. Observe that $\widetilde{N}_{\widetilde{S}}$ is a submatrix of $N'_{S'}$, therefore it is a jumbled submatrix of N_S. By Observation 3, $\mathsf{rank}(\widetilde{N}_{\widetilde{S}}) \leq \mathsf{rank}(N_S) \leq r$, hence $(\widetilde{N}, r, k)$ is a Yes-instance of Matrix Rigidity.

In the backward direction, suppose $(\widetilde{N}, r, k)$ is a Yes-instance of Matrix Rigidity. If the output of $\widetilde{N}$ occurs at lines 1 or 4, then we actually know the solution to the instance (N, r, k) of Matrix Rigidity as explained in the comment of the pseudocode. If the output occurs at line 6c, then the output $\widetilde{N}$ of Column-Reduction is a jumbled matrix of N and the result holds by Observation 2. Now we are left with the case when the output occurs at line 9. Let $\widetilde{S}$ be any solution to the instance $(\widetilde{N}, r, k)$ of Matrix Rigidity. The matrix edited using a solution $\widetilde{S}$ is denoted by $\widetilde{N}_{\widetilde{S}}$. Notice that the matrix $\widetilde{N}$ consists of two submatrices $\mathcal{L}$ and $\mathcal{B}$. As $\mathcal{L}$ consists of $i_{\leq r}$ blocks having rank $r + 1$, by Observation 1, we need to edit at least $i_{\leq r}$ entries in $\mathcal{L}$. So, we can afford to make at most $k - i_{\leq r}$ edits in the matrix $\mathcal{B}$. As $\mathcal{B}$ consists of $k + 1 - i_{\leq r}$ blocks, by pigeonhole principle there exists at least one block in $\mathcal{B}$, say B_t, which is not subject to any edit by the solution $\widetilde{S}$. Construct the matrix N' by concatenating the columns of M_{k+1} (the columns discarded by the procedure) at the end of the matrix $\widetilde{N}$. By construction N' is a jumbled matrix of N and $\widetilde{N}$ is its submatrix. Moreover, the matrix $N'_{\widetilde{S}}$ has rank at most r due to the presence of the unedited block B_t in $\widetilde{N}_{\widetilde{S}}$ which spans the matrix M_{k+1}. As $\widetilde{S}$ is a solution of (N', r, k), use Lemma 28 to get a solution S of (N, r, k). Thus, $\mathsf{rank}(N_S) = \mathsf{rank}(N'_{\widetilde{S}}) = \mathsf{rank}(\widetilde{N}_{\widetilde{S}}) \leq r$, proving that the

instance (N, r, k) is a YES-instance of MATRIX RIGIDITY.

To complete the proof, observe that in the procedure MATRIX-REDUCTION, the instances (A, r, k) and (C_A, r, k) are equivalent by the argument above. By Observation 2, (C_A, r, k) and (C_A^T, r, k) are equivalent. As R_A is the output of COLUMN-REDUCTION, (R_A, r, k) is equivalent to (C_A, r, k). Finally, by Observation 2, again (R_A, r, k) and (R_A^T, r, k) are equivalent. □

If the matrix A is over a fixed finite field $\mathbb{F}$, we obtain a kernel as well.

Theorem 26. *Given an instance* (A, r, k) *of* FF MATRIX RIGIDITY *over the field* $\mathbb{F}_p$*, the procedure* MATRIX-REDUCTION *outputs an* $O(r^2 \cdot k^2 \cdot \log p)$*-kernel.*

Proof. The number of entries in the output matrix of MATRIX-REDUCTION is bounded by $O(r^2 \cdot k^2)$, and the bit length of each entry is at most $\lceil \log_2 p \rceil$. □

In case the field $\mathbb{F}$ is infinite—for example, if $\mathbb{F}$ is either $\mathbb{Q}$ or $\mathbb{R}$—the procedure is not guaranteed to produce a kernel as the bit lengths of matrix entries may not be bounded by a function of r and k.

4.4 Fixed-Parameter Tractability with Respect to $k + r$

This section describes an algorithm for MATRIX RIGIDITY. The formulation it presents was also used in the context of complexity analysis in [95].

Using Lemma 29, we can reduce any instance (A, r, k) to an equivalent instance (A', r, k) such that the matrix A' is a jumbled submatrix of A and the number of entries in A' is $O(r^2 \cdot k^2)$. Once we have such a matrix A', it is useful to examine an alternative definition of the rank of a matrix, which is given in terms of the determinant of its square submatrices. Specifically, we will rely on the following proposition. We include a proof for the sake of completeness.

Proposition 1 (see Chapter 7 in [98]). *A matrix A over $\mathbb{R}$ has rank at most r if and only if all the $(r+1)\times(r+1)$ submatrices of A have determinant 0.*

The correctness of our algorithm MatRig-Alg for Matrix Rigidity, which is described in Figure 4.3, follows in a straightforward fashion using Proposition 1. This algorithm for Matrix Rigidity crucially relies on a procedure which can decide the feasibility of a system of polynomials over a given field. This procedure shall be the object of discussion in the rest of the section.

Observe that each polynomial in $\mathcal{P}$, as defined in the algorithm MatRig-Alg, has at most k unknowns and its *total degree* is at most k. The size of $\mathcal{P}$ is of order $(r\cdot k)^{O(r)}$. The bit sizes of the coefficients of polynomials in $\mathcal{P}$ are bounded using the following.

Lemma 30. *Let A be a matrix over $\mathbb{R}$. If the longest length entry in A has bit length L then the bit lengths of the coefficients of the polynomials in $\mathcal{P}$, as computed by the algorithm* MatRig-Alg, *are of size $O(r\cdot L + r\cdot\log r)$.*

Proof. The coefficients of polynomials in $\mathcal{P}$ are obtained by computing the determinant of matrices which have size at most $r\times r$. Moreover, the coefficient of a monomial is given by the determinant of a single matrix (as opposed to being the sum of many determinants) because the indeterminates occur only once in the *mixed matrix*. By *Hadamard's inequality* (for a proof see [69]), for a $r\times r$ matrix M, we have $\det(M)\leq\prod_{i\in[r]}\|M_i\|_2$. As the bit length of entries in A is at most L, the coefficients of polynomials in $\mathcal{P}$ are at most $\prod_{i\in[r]}\sqrt{r\cdot 2^{L+1}} = (r\cdot 2^{L+1})^{\frac{r}{2}}$; taking its logarithm gives us the bit length. □

We use the following proposition to check the feasibility of the system of polynomials $\mathcal{P}$ when it is defined over $\mathbb{R}$.

Proposition 2 (see Proposition 4.2 in [93]). *Given a set $\mathcal{P}$ of ℓ polynomials of degree d in k variables with integer coefficients of bit length L, we can decide the feasiblility of $\mathcal{P}$ with $L\log L\log\log L(\ell\cdot d)^{O(k)}$ bit operations.*

Applying the proposition above on the system of equations $\mathcal{P}$, we get the following.

Theorem 27. *Given a matrix A over $\mathbb{R}$ such that the bit length of each of its entries is bounded by L, and let r and k be two non-negative integers. Then, the instance (A, r, k) of* Real Matrix Rigidity *can be solved in time $O^*(2^{O(r \cdot k \cdot \log(r \cdot k))})$.*

Proof. The algorithm MatRig-Alg generates $O((r \cdot k)^{2k})$ systems of equations. Each system of equations has $\ell = (r \cdot k)^{O(r)}$ equations, where the degree $d = k$ and there are k variables. Using Proposition 2 along with Lemma 30, we get the required running time.

Notice that a system of equations $\mathcal{P}$ is feasible if and only if the chosen entries of the matrix can be edited to reduce the rank. Since we exhaustively try all possible entries that can be edited, the correctness of MatRig-Alg follows. □

In the case where the underlying field $\mathbb{F}_p$ is finite, the coefficients of the polynomials are elements of $\mathbb{F}_p$ and hence have bounded bit lengths. The feasibility of $\mathcal{P}$ over a finite field can be decided using the following known algorithm which also gives us an algorithm for FF Matrix Rigidity.

Proposition 3 (Kayal [62])**.** *There is a deterministic algorithm which, given an input consisting of a finite field $\mathbb{F}_p$ and system of polynomials $f_1, \ldots, f_\ell \in \mathbb{F}_p[x_1, \ldots, x_k]$ of* total degree *bounded by d, decides its feasibility in time $d^{k^{O(k)}} \cdot (\ell \cdot \log p)^{O(1)}$.*

Similar to the proof of Theorem 27, we obtain the following.

Theorem 28. *The problem* FF Matrix Rigidity*, where the input matrix A is an $m \times n$ matrix over a field $\mathbb{F}_p$, can be solved in time $f(r, k)(\log p + m + n)^{O(1)}$ for some function f.*

This algorithm for FF Matrix Rigidity has the advantage that it runs in time which is *polynomial in the logarithm of the order of the field*, even though the dependence on k is exponential.

4.5 W[1]-Hardness with Respect to k

In this section, we first reduce (in two steps) a special case of Odd Set to a problem that has a formulation easier to use in our context. The latter problem is reduced to a variant of Nearest Codeword, which, in turn, is reduced to Real Matrix Rigidity and FF Matrix Rigidity.

Odd Set	**Parameter:** k
Input: A family $\mathcal{F}$ of sets over a universe U and a non-negative integer k.	
Question: Does there exist a subset $S \subseteq U$ of size at most k such that the intersection of S with every set in $\mathcal{F}$ has odd size?	

Nearest Codeword	**Parameter:** k
Input: An $m \times n$ matrix M and an m-dimensional vector b over $\mathbb{F}_2$, along with a non-negative integer k.	
Question: Is there an n-dimensional vector x over $\mathbb{F}_2$ such that the Hamming distance between Mx and b is at most k?	

Theorem 29. Real Matrix Rigidity *and* FF Matrix Rigidity *for any choice of a finite field* $\mathbb{F}_p$ *are* W[1]*-hard with respect to* k*.*

Proof. Let $\mathbb{F}$ denote the field, which is either $\mathbb{R}$ or some finite field $\mathbb{F}_p$, over which we define Matrix Rigidity. First, we observe that the reduction from Multicolored Clique given in the book [27] (Theorem 13.31) to show that Odd Set is W[1]-hard actually shows that the following special case of Odd Set is W[1]-hard. That is, the constructed instances have the form specified in the special case.

Partitioned Odd Set	**Parameter:** k
Input: A family $\mathcal{F}$ of sets over a universe U, a non-negative integer k, a partition $(U_1, \ldots, U_k)$ of U such that for every $i \in [k]$, $U_i \in \mathcal{F}$, and for every $F \in \mathcal{F}$, there exist $i, j \in [k]$ for which $F \subseteq U_i \cup U_j$.	
Question: Is there a subset $S \subseteq U$ of size at most k such that the intersection of S with every set in $\mathcal{F}$ has size 1?	

The arguments below will crucially rely on the fact that we restrict ourselves to this special case. Given a vector v, we let $\text{supp}(v)$ denote the indices of the entries of v that do not contain 0. Now, we reformulate Partitioned Odd Set in the language of matrices as follows.

Partitioned Odd Matrix	**Parameter:** k
Input: A $t \times r$ binary matrix L over $\mathbb{R}$, a non-negative integer k, a partition $(U_1, \ldots, U_k)$ of $[r]$ such that for every $i \in [k]$, there exists $j \in [t]$ for which $U_i = \text{supp}(L_j)$, and for every $i \in [t]$, there exist $j, \ell \in [k]$ for which $\text{supp}(L_i) \subseteq U_j \cup U_\ell$.	
Question: Is there an r-dimensional binary vector x such that $\lvert\text{supp}(x)\rvert \leq k$ and $Lx = 1$?	

Given an instance $(\mathcal{F}, U, (U_1, \ldots, U_k), k)$ of Partitioned Odd Set, it is straightforward to obtain (in polynomial time) an equivalent instance $(L_{t\times r}, (U'_1, \ldots, U'_k), k')$ of Partitioned Odd Matrix as follows. First, we let $t = |\mathcal{F}|$ and $r = |U|$. We assume w.l.o.g. that $U = [r]$. Now, we associate a row L_i with each set $F \in \mathcal{F}$ by letting L_i contain 1 at each entry whose index belongs to F and 0 at each of the remaining entries. That is, $\text{supp}(L_i) = F$. Finally, we let $(U'_1, \ldots, U'_k) = (U_1, \ldots, U_k)$ and $k' = k$. It is easy to see that $S \subseteq U$ is a solution to $(\mathcal{F}, U, (U_1, \ldots, U_k), k)$ if and only if the binary vector $x_{r\times 1}$ such that $\text{supp}(x) = S$ is a solution to $(L_{t\times r}, (U'_1, \ldots, U'_k), k)$, and therefore the instances are equivalent.

We now incorporate the input field $\mathbb{F}$.

$\mathbb{F}$-Odd Matrix **Parameter:** k

Input: A $t \times r$ binary matrix L over $\mathbb{F}$ and a non-negative integer k.

Question: Is there an r-dimensional vector x over $\mathbb{F}$ such that $|\text{supp}(x)| \leq k$ and $Lx = 1$?

We reduce Partitioned Odd Matrix to $\mathbb{F}$-Odd Matrix as follows. Given an instance $(L_{t\times r}, (U_1, \ldots, U_k), k)$ of Partitioned Odd Matrix, we simply output $(L_{t\times r}, k)$ as the equivalent instance of $\mathbb{F}$-Odd Matrix. In one direction, let x be a solution to $(L_{t\times r}, (U_1, \ldots, U_k), k)$. Recall that L is a binary matrix. Thus, since x is a binary vector satisfying $Lx = 1$ over $\mathbb{R}$, it must also satisfy $Lx = 1$ over $\mathbb{F}$. Since $|\text{supp}(x)| \leq k$, we get that x is a solution to $(L_{t\times r}, k)$. In the second direction, let x be a solution to $(L_{t\times r}, k)$. Assume w.l.o.g. that for $s \in [k]$ we have $\text{supp}(L_s) = U_s$. As the given k sets U_i form a partition and $|\text{supp}(x)| \leq k$, for every $s \in [k]$ we have $|\text{supp}(x) \cap U_s| = 1$. Since L is a binary matrix and for every $s \in [k]$ we have $L_s x = 1$ over $\mathbb{F}$, it implies that x is a binary vector. It remains to show, that $Lx = 1$ over $\mathbb{R}$. For any index $i \in [t]$, there exist $j, \ell \in [k]$ such that $\text{supp}(L_i) \subseteq U_j \cup U_\ell$. As L and x are both binary, over $\mathbb{R}$ we have $1 \leq L_i x \leq L_j x + L_\ell x \leq 2$. To complete the proof, we claim that $L_i x \neq 2$ over $\mathbb{R}$. Assume that $\mathbb{F} = \mathbb{F}_p$ i.e. the order of field is some prime number p. On the contrary, assume that $L_i x = 2$ over $\mathbb{R}$. For $L_i x$ to equal 1 over $\mathbb{F}$, we must have $(2 \mod p) = 1$, which is impossible as $p \geq 2$. This allows us to conclude that $L_i x = 1$ also over $\mathbb{R}$.

In what follows, calculations are performed over $\mathbb{F}$. Next, we reduce $\mathbb{F}$-Odd Matrix to the following variant of the Nearest Codeword problem which is inspired by a reduction from Nearest Codeword to Odd Set of Bonnet et al. [12].

$\mathbb{F}$-Nearest Codeword **Parameter:** k

Input: An $m \times n$ matrix M, an m-dimensional vector b over $\mathbb{F}$, and a non-negative integer k.

Question: Is there an n-dimensional vector y over $\mathbb{F}$ such that the Hamming distance between My and b is at most k?

Given an instance $(L_{t\times r}, k)$ of $\mathbb{F}$-Odd Matrix, construct an instance $(M_{m\times n}, b, k')$ of $\mathbb{F}$-Nearest Codeword as follows. First, let $k' = k$. Now, let M be an $m \times n$ matrix, where $m = r$ and $n = r - \mathsf{rank}(L)$, such that the rows of L form a basis for the subspace orthogonal to the column space of M. Then, an r-dimensional vector v over $\mathbb{F}$ satisfies $Lv = 0$ if and only if v belongs to the column space of M (i.e., there is an n-dimensional vector y over $\mathbb{F}$ such that $My = v$). Finally, let b be an r-dimensional vector such that $Lb = -1$. If no such vector exists, then there is no r-dimensional vector over $\mathbb{F}$ such that $Lv = 1$, which in particular implies that $(L_{t\times r}, k)$ is a No-instance, and thus we can return a trivial No-instance of $\mathbb{F}$-Nearest Codeword. Therefore, next assume that b exists. To prove that the reduction is correct, first let x be a solution to $(L_{t\times r}, k)$. Then, $Lx = 1$, and since $Lb = -1$, we have that $L(x+b) = Lx + Lb = Lx - 1 = 0$. Therefore, by the choice of M, there exists an n-dimensional vector y over $\mathbb{F}$ such that $My = (x+b)$. Since $|\mathrm{supp}(x)| \leq k$, we have that the Hamming distance between My and b is at most k, which implies that y is a solution to $(M_{m\times n}, b, k')$. In the other direction, let y be a solution to $(M_{m\times n}, b, k')$. Then, since the Hamming distance between My and b is at most k, there exists an m-dimensional vector x such that $|\mathrm{supp}(x)| \leq k$ and $My = x + b$. Therefore, by the choice of M, $L(x + b) = 0$. Since $Lb = -1$, we get that $Lx = 1$, which implies that x is a solution to $(L_{t\times r}, k)$.

Finally, we reduce $\mathbb{F}$-Nearest Codeword to Matrix Rigidity over $\mathbb{F}$. For this purpose, let $(M_{m\times n}, b, k)$ be an instance of $\mathbb{F}$-Nearest Codeword. We can assume that the columns of M are linearly independent. To see this, let $n' = \mathsf{rank}(M)$ and let M' be the $m \times n'$ submatrix of M whose columns are a column basis of M. Notice that the span of columns of M and M' are exactly the same. For any choice of vector y, the vector My lies in the column space of M'. Thus it easily follows that the instances (M, b) and (M', b) are equivalent instances of $\mathbb{F}$-Nearest Codeword. Therefore, for the rest of the proof we assume w.l.o.g. that the columns of M are linearly independent. We construct an equivalent instance $(A_{s\times t}, r, k)$ of Matrix Rigidity over $\mathbb{F}$ as follows. Let $s = m$, $r = n$ and

$t = (k+1)n+1$. The matrix A consists of $k+1$ repeated copies of M and b as the last column:

$$A = [\underbrace{M, \ldots, M}_{k+1 \text{ times}}, b].$$

On the one hand, let y be a solution to $(M_{m\times n}, b, k)$. Then, there are at most k entries that should be changed in b to obtain an m-dimensional vector b' over $\mathbb{F}$ such that $My = b'$. In the matrix A, replace the last column b by b'. Denote the resulting matrix by A'. Then, the last column of A' is a linear combination of its other columns, by the construction of A and since $My = b'$. Therefore, $\mathsf{rank}(A') = n$, which implies that $(A_{s\times t}, r, k)$ is a Yes-instance. In the other direction, suppose that $(A_{s\times t}, r, k)$ is a Yes-instance. Then, it is possible to change at most k entries in A and obtain a matrix A' such that $\mathsf{rank}(A') = n$. Besides the last column of A, A consists of $k + 1$ repeated copies of M (i.e., more times than the number of changes), it must be that one copy of M remains unedited in A'. Let b' be the last column of A', as the rank of A' is n, we get that there exists an n-dimensional vector y over $\mathbb{F}$ such that $My = b'$. Since the Hamming distance between b and b' is at most k, we have that y is a solution to $(M_{m\times n}, b, k)$. □

4.6 An Algorithm for FF Matrix Rigidity with Subexponential Dependency on k

In this section, we will also rely on the classic technique of bounded search trees, which is presented in the Preliminaries. Our objective is to prove the following theorem.

Theorem 30. *For some function f, the* FF Matrix Rigidity *problem is solvable in $O^*(2^{O(f(r,p)\sqrt{k}\log k)})$ time.*

Let $\mathbb{F}_p$ be a finite field. We will prove that Matrix Rigidity over $\mathbb{F}_p$ is solvable in the desired time. Let $(A_{m\times n}, r, k)$ be an instance of this problem. In Theorem 14 of Chapter 3,

we proved that any rank r skew-symmetric matrix A with entries from $\{-1, 0, 1\}$ has at most 3^r distinct columns. That proof with a straightforward modification gives us the following corollary which we state here without proof.

Corollary 10. *Any rank r symmetric matrix A with entries from $\mathbb{F}_p$ has at most p^r distinct rows and at most p^r distinct columns.*

Given a matrix M, let I be a maximum sized set of distinct rows of M, and let J be a maximum sized set of distinct columns of M. Then, we define $\mathsf{distinct}(M) = M[I, J]$. To be more precise, $\mathsf{distinct}(M)$ should be defined as an equivalence class of submatrices up to reordering of rows and columns, but here we slightly abuse notation and consider some specific submatrix $M[I, J]$ as $\mathsf{distinct}(M)$. Now, let $\mathcal{D}_r$ be the set of all rank r matrices with distinct rows and distinct columns. As each matrix in $\mathcal{D}_r$ has at most p^r rows as well as at most p^r columns, we get the following observation.

Observation 8. *The value of $|\mathcal{D}_r|$ is bounded by a function of r and p.*

To solve $(A_{m\times n}, r, k)$ in the desired time, for each $D \in \mathcal{D}_r$, we need to check in time $O^*(2^{O(f(r,p)\sqrt{k}\log k)})$, for some function f, whether it is possible to change at most k entries of A to obtain a matrix M such that $\mathsf{distinct}(M) = D$.

We would be reducing this problem into a graph problem. Next, we show how to interpret a given matrix as the adjacency matrix of a weighted undirected graph. Given an $m \times n$ matrix M, let

$$\mathsf{sym}(M) = \begin{bmatrix} \mathbf{0} & M \\ M^{\mathrm{T}} & \mathbf{0} \end{bmatrix},$$

where $\mathbf{0}$ is the matrix of appropriate dimension that contains zero at each of its entries. Now, suppose that we are given a matrix $D \in \mathcal{D}_r$. Observe that at most k entries in A can be changed to obtain a matrix M such that $\mathsf{distinct}(M) = D$ if and only if there are at most k pairs (i, j), $1 \leq i \leq m$ and $1 \leq j \leq n$, such that the entries $a_{i,j+m}$ and $a_{i+n,j}$ in $\mathsf{sym}(A)$ can be changed to obtain a matrix M such that $\mathsf{distinct}(M) = \mathsf{sym}(D)$. Now, we think of

the matrices $\mathsf{sym}(A)$ and $\mathsf{sym}(D)$ as adjacency matrices of weighted undirected complete graphs where the weights belong to $\mathbb{F}_p$. More precisely, given a symmetric matrix $M_{t\times t}$ with zeros at its diagonal, the construction of $\mathsf{graph}(M)$ is performed as follows. For each index $i \in [t]$ we introduce a vertex v_i, and the weight of an edge $\{v_i, v_j\}$ is given at the entry m_{ij}. Given a weighted undirected complete graph $G = (V, E)$, let $(V_1, \ldots, V_\ell)$ be a partition of V minimizing ℓ such that for all $i \in [\ell]$, the weight of each edge between any two vertices in V_i is 0, and for all distinct $i, j \in [\ell]$, $v, v' \in V_i$ and $u \in V_j$, the weight of $\{v, u\}$ equals the weight of $\{v', u\}$. Observe that this partition is unique up to reordering the sets V_i. Informally, two vertices of the graph are in the same partition if they correspond to entrywise equal columns. Now, we let $\mathsf{distinct}(G)$ be the weighted undirected complete graph having one vertex representing each set V_i, and letting the edge between the vertex representing V_i and the vertex representing V_j have the same weight as any edge between a vertex in V_i and a vertex in V_j in G. Since changing an entry in a matrix M is equivalent to changing the weight of an edge in $\mathsf{graph}(\mathsf{sym}(M))$, we conclude that to prove Theorem 30, it is sufficient to prove the following lemma.

Lemma 31. *Let G and H be weighted undirected complete graphs with weights from $\mathbb{F}_p$ such that $\mathsf{distinct}(H) = H$ and $|V(H)| = g(r, p)$ for some function g. Then, it is possible to determine in time $O^*(2^{O(f(r,p)\sqrt{k}\log k)})$, for some function f, whether the weights of at most k edges in G can be changed to obtain a graph G' such that $\mathsf{distinct}(G') = H$.*

Thus, in the rest of this section, it is sufficient to focus on the proof of Lemma 31, which is based on the proof of Theorem 21 given in Chapter 3 (which, in turn, is inspired by the paper [29]). The proof idea relies on a branching algorithm in which the branching parameter is the number of edits allowed for a particular branch. The proof transforms a given graph G into another fixed graph H, satisfying $\mathsf{distinct}(H) = H$, whose rank is already known to be at most r, the target rank. To do so, the vertices of H are treated like bags which need to be filled in with vertices of G. The operation of placing a vertex inside a bag corresponds to creating a repeated row and a column, which does not change

the rank. Finally, we need to keep track of number of edits required for a vertex to be placed in a particular bag of H. This procedure ensures that the graph G is step by step transformed into another graph G' such that $\mathsf{distinct}(G')$ is a subgraph of H, thus the rank of G' is at most the rank of H. Next, we state a lemma about the branching vectors.

Proposition 4 (see [29]). *For $t > 0$, let $(1, t, t, \ldots, t)$ be the branching vector in which t appears s times. If t is significantly larger than s ($t > 2^s$), then its branching number is bounded by $1 + \frac{\log_2 t}{t}$.*

Proof of Lemma 31. A vertex of H will be referred to as a bag and will be filled in with the vertices of G. Let $b = |V(H)|$ denote the number of bags in H and use $\mathcal{B} = \{B_i : i \in [b]\}$ to denote the set of bags. Assume that the vertices in H are $v_1, \ldots, v_b$ and B_i corresponds to the vertex v_i, for all $i \in [b]$. The collection of vertex-sets of G corresponding to the vertices in $\mathsf{distinct}(G)$ is denoted by $\mathcal{M} = \{V_1, \ldots, V_t\}$ (recall that $\mathcal{M}$ is a partition of $V(G)$ and each V_i denotes a set of vertices in G represented by the same vertex in $\mathsf{distinct}(G)$). Observe that each change of a weight of an edge in G can decrease the number of vertices in $\mathsf{distinct}(G)$ by at most 2, and therefore if $t > 2k + |V(H)|$, the answer to (G, H, r, p, k) is No. Therefore, we may next assume that $t \leq 2k + |V(H)|$. If at any point during the calculations below, the value of the parameter k drops below 0, we return the answer No at the current node of the search tree. If at least one leaf of the search tree returns Yes, we propagate the value Yes – that is, if a node has several children (corresponding to different branches considered by a branching rule) and at least one of them returns Yes, we return Yes. In the preprocessing phase of the algorithm, we add a "sufficient" number of vertices to each bag, which will allow us later to perform branching rules associated with branching vectors where the drop in the parameter at all but one branch is large.

Preprocessing phase: Each bag can be in one of two states: *closed* or *open*. At the start of the algorithm all the bags are empty and open. In every bag we create $s = \lfloor \sqrt{k} \rfloor$ empty slots. Overall there are $b \cdot s$ slots that can be filled. We say that a bag is *free* if it is open and less than s slots have been filled in it. The state of a node in the bounded search tree is

denoted using $(\mathcal{M}, \mathcal{B})$. As long as there is a free bag, we perform the following branching rule, consisting of $t + 1$ branches. In its i^{th} branch, for $i \in [t]$, pick an arbitrary vertex u from V_i, delete it from V_i and add it to the lowest number free bag in $\mathcal{B}$. In the $(t + 1)^{th}$ branch we mark the lowest number free bag as closed. The application of branching rules in the preprocessing phase is finished when no bag is free. At each leaf node $(\mathcal{M}, \mathcal{B})$ of the search tree, we construct a graph H' over the vertices in $\bigcup \mathcal{B}$. For every weighted edge $\{v_i, v_j\}$ in H we add all the edges $B_i \times B_j$ in the graph H' with the same weight as $\{v_i, v_j\}$. Edges between vertices of the same bag have weight 0. Next, for every edge $\{v, u\}$ in H', we check whether the weight of the edge in H' is the same as its weight in G – if this is not the case, we decrease k by 1. This operation is equivalent to changing the weight of the edge in G to its weight in H'.

At the end of the preprocessing phase the following two cases arise. For each vertex v_i of H either we know exactly which are the at most s vertices of G that should be equivalent to it in a solution graph G' (i.e., $\mathsf{distinct}(G') = H$), or we know exactly s vertices of G equivalent to v_i in G'. In the first case, the bag has been closed, and in the second case, it remains open. In the preprocessing phase, each branching rule consists of at most $t + 1$ branches and the depth of the search tree is $b \cdot s$. This gives us the following.

Observation 9. *The preprocessing phase can be performed in time* $O^*((t+1)^{b \cdot s}) = O^*((b+2k+1)^{b \cdot s}) = O^*(2^{O(f'(r,p)\sqrt{k}\log k)})$ *for some function* f'.

Assigning bags to undecided vertices: This phase of the algorithm begins at a node $(\mathcal{M}, \mathcal{B})$ along with the graph H' and the reduced parameter k as provided by a leaf of the search tree procedure in the previous phase. The vertices in $V(G) \setminus (\bigcup \mathcal{B})$, which have not yet been added to any bag, are called *undecided vertices*. We note that so far the weight modifications have been done only within the bag vertices added in the preprocessing phase, and that the (possibly modified) weights of edges between these bag vertices remain fixed for the rest of the algorithm. The branching rules stated in the next paragraph are applied exhaustively in the given order – if at any node of the search tree a rule is applied, then

none of the previous rules is applicable. *We first consider the case when all the bags are open and handle the closed bags later.*

Before an undecided vertex is placed in a bag, as a first step we need to ensure that its edge weight with all the vertices of particular bag are the same. If there exists an undecided vertex u and a bag $B_i \in \mathcal{B}$ such that not all of the edges between u and the vertices in B_i have the same weight then we apply the following rule. For each weight in $\mathbb{F}_p$, we have a separate branch. In the branch corresponding to some weight $w \in \mathbb{F}_p$, for each edge between u and a vertex in B_i whose weight is not w, we change the weight of the edge to w and decrease k by 1. For now, u is not yet added to any bag, and remains an undecided vertex. Let us denote $\mathbb{F}_p = \{w_1, w_2, \ldots, w_p\}$. Then, for all $j \in [p]$, we let s_j denote the number of edges between u and B_i whose weight is not $w_j \in \mathbb{F}_p$. Let us denote $\ell = |B_i|$. Notice that $\sum_{j=1}^{p} s_j = \ell$. As B_i is an open bag (due to our current assumption), it has at least s slots filled, which means that $\ell \geq s$. Observe that the branching vector we obtain is precisely $(\ell - s_1, \ell - s_2, \ldots, \ell - s_p)$.

We now show that the branching number of $(\ell - s_1, \ell - s_2, \ldots, \ell - s_p)$ is upper bounded by $1 + \frac{\log_2(s/2)}{(s/2)}$. For this purpose, since $\ell \geq s$, it is sufficient to show that the branching number of $(\ell - s_1, \ell - s_2, \ldots, \ell - s_p)$ is upper bounded by $1 + \frac{\log_2(\ell/2)}{(\ell/2)}$. As $\sum_{j=1}^{p} s_j = \ell$, there can be at most one $j \in [p]$ such that $s_j > \ell/2$. Without loss of generality, suppose that this j equals 1. Then, $(\ell - s_1, \ell - s_2, \ldots, \ell - s_p)$ is at least as good as $(1, \ell - s_2, \ldots, \ell - s_p)$, where each s_j is at most $\ell/2$. In turn, this means that the latter branching vector is at least as good as $(1, \ell/2, \ell/2, ..., \ell/2)$, where $\ell/2$ occurs $(p-1)$ times. Then, by Proposition 4, we derive that the root of this branching vector is upper bounded by $1 + \frac{\log_2(\ell/2)}{(\ell/2)}$. Now, after applying this rule exhaustively, given any undecided vertex u, for each bag in $\mathcal{B}$ the weights of the edges between u and the vertices in this bag are the same.

We say that a vertex u fits a bag $B(u) \in \mathcal{B}$, if u has edges of the same weights as vertices in $B(u)$ in the graph induced on the bags. That is, the weight of each edge between u and a vertex in $B(u)$ is 0, and the weight of an edge between u and a vertex $v \in (\bigcup \mathcal{B}) \setminus B(u)$ is the

same as the weight of an edge between any vertex in $B(u)$ and v. Since $\mathsf{distinct}(H) = H$, all the vertices in H have distinct weighted neighborhoods, and therefore every vertex u fits at most one bag. If there exists an undecided vertex u that fits no bag, we branch and decide on a bag for u, put u in this bag, and perform the necessary changes of weights of edges between u and vertices in $\bigcup \mathcal{B}$, updating k accordingly. Here, we have b branches, and at each branch the weights of all of the edges between u and at least one bag are changed, and therefore the parameter decreases by at least s. That is, we obtain a branching vector at least as good as $(s, \ldots, s)$, where s appears b times. Below we will obtain a worse branching vector.

After the last step, every undecided vertex u fits exactly one bag $B(u)$. If there are no two undecided vertices u and v such that if we put u in $B(u)$ and v in $B(v)$, no conflict is created (that is, the weight of the edge between u and v is the same as the weight of any edge between a vertex in $B(u)$ and a vertex in $B(v)$), we can simply return the answer Yes. Indeed, in this case, since $k \geq 0$ (else recall that we would have already returned No), we have used at most k changes to modify G to a graph G' such that $\mathsf{distinct}(G') = H$ – each undecided vertex can be put in the only bag it fits and no changes are required. Therefore, we now suppose that there are two undecided vertices u and v that do create a conflict. We apply a branching rule that is an exhaustive search consisting of the following branches:

1. In the first branch, we address the conflict by changing the weight of the edge between u and v to the weight of any edge between a vertex in $B(u)$ and a vertex in $B(v)$, decrease k by 1, and then put u in $B(u)$ and v in $B(v)$.

2. Next, we consider $b - 1$ branches that find a new bag for u. More precisely, in each of these branches, we put u in a different bag B_i in $\mathcal{B} \setminus \{B(u)\}$, update the weights of the edges between u and other vertices in $\bigcup \mathcal{B}$ accordingly (that is, if the weight of an edge between u and a vertex in B_i is not 0, we update it to 0, and if the weight of an edge between u and a vertex in a different bag B_j is not the same as the weight of the edge $\{v_i, v_j\}$ in H, we update it to this weight). Moreover, in each of these

branches, we decrease k by the number of the changes that were made. Observe that since u fits only $B(u)$, in each of these branches k is decreased by at least s.

3. Finally, we consider $b - 1$ branches that find a new bag or v. These branches are symmetric to those considered in the previous item.

The branching vector we obtain is at least as good as $(1, s, \ldots, s)$, where s appears $2(b-1)$ times. By Proposition 4, the branching number is bounded by $1 + \frac{\log_2 s}{s}$.

Thus, if there are no closed bags the branching rules mentioned above will be sufficient to decide the fate of that branch. Next, we will show that closed bags can be ignored safely during the branching procedure and the branching rules can be applied on the graph induced on open bags.

Handling the closed bags: The closed bags do not guarantee branching vectors as good as those given previously. For example, when we change all of the weights of the edges between some undecided vertex u and the vertices of a closed bag, we are changing less than s values, and therefore the drop in k is smaller than s. However, the closed bags do not really pose a problem due to the following arguments, which rely on the fact that after a bag is marked closed in the preprocessing phase, no vertex will ever be added in it later. In what follows, we show that we can handle the closed bags by making greedy choices after the branching choices have been made according to the graph induced on the open bags.

Let U be the set of vertices of H' corresponding to the *open* bags. Note that $H'[U]$ may not remain a graph satisfying $\mathsf{distinct}(H'[U]) = H'[U]$. In that case we group together bags which are equivalent in $H'[U]$ and call each group a superbag, where two bags B_i an B_j are equivalent if the weight of each edge between them is 0, and for every bag B_ℓ, the weight of an edge between a vertex in B_i and a vertex in B_ℓ is the same as the weight any edge between a vertex in B_j and a vertex in B_ℓ. So each bag in $H'[U]$ is either a "normal" bag or a superbag. Observe that each bag in $H'[U]$ has at least s vertices in it as we are only merging open bags together, which had s vertices in them to begin with.

Considering the graph $H[U]$ with the superbags, we perform exactly the same branching rules as above, where we assume that all the bags are open. We have fewer branches and the branching vectors do not get worse. After branching, at the point where we have previously returned YES, we have that the each undecided vertex fits one of the bags in $H'[U]$ and the weights of the edges between them are fixed without conflicts. Now, while actually adding a vertex u to a bag we also decide on the bag within a superbag S that will host u. Adding u to a bag does not change the weights of edges within the union of open bags and set of undecided vertices. Therefore we can take these decisions independently for each u, adding u to any bag in S that causes the minimum number of changes of weights of edges between u and vertices in the closed bags, and updating k accordingly.

Time complexity: Recall that the preprocessing phase is performed in the desired time. Thus, it remains to analyze the time necessary to perform the calculations following this phase. The worst branching number we obtained was bounded by $1 + \frac{\log_2(s/2)}{(s/2)}$, assuming that s is significantly larger than r and p. Therefore, since $s = \lfloor \sqrt{k} \rfloor$, we obtain that the running time of our algorithm is bounded by

$$O^*\left(2^{f(r,p)} \cdot \left(1 + \frac{\log_2(s/2)}{(s/2)}\right)^k\right) = O^*(2^{O(f(r,p)\sqrt{k}\log k)})$$

for some function f. □

Algorithm: COLUMN-REDUCTION
INPUT: A matrix A over some field $\mathbb{F}$, and two non-negative integers r, k.
OUTPUT: A matrix having $O(r \cdot k)$ columns.

1. **if** $\mathsf{rank}(A) \leq r$ **then** return a YES-instance of appropriate size and exit.
2. Initialize $M_0 = A$ and $i = 0$.
3. **while** $\mathsf{rank}(M_i) \geq r + 1$:
   (a) Let L_i be a set of columns of M_i which is linearly independent in $\mathbb{F}$ and whose size is $r + 1$.
   (b) Let M_{i+1} be the matrix obtained by deleting the columns in L_i from M_i.
   (c) Increment i by 1.
4. **if** $i > k$ **then** return a No-instance of appropriate size and exit.

   // The matrix A has more than k pairwise-disjoint blocks of the form L_j for $j \leq i$, each having $r + 1$ linearly independent columns. By Observation 1, each block L_i requires at least 1 edit, hence, by Observation 4, (A, r, k) is a No*-instance of* MATRIX RIGIDITY.
5. Let $i_{\leq r} = i$ store the index where the rank of M_i falls below $r + 1$.
6. **while** $i \leq k$:
   (a) Let B_i be a column basis of M_i.
   (b) Obtain M_{i+1} by deleting the columns in B_i from M_i.
   (c) **if** M_{i+1} is empty (in other words, $B_i = M_i$) **then** return A.
   (d) Increment i by 1.
7. Let $\mathcal{L}$ be a matrix formed by the columns in each L_i for $i \in \{0, \ldots, i_{\leq r} - 1\}$.
8. Let $\mathcal{B}$ be a matrix formed by the columns in each B_i for $i \in \{i_{\leq r}, \ldots, k\}$.
9. Return the matrix formed by the columns in $\mathcal{L} \cup \mathcal{B}$.

   //Note that M_{k+1} is non-empty if output occurs here.

Figure 4.1: The column reduction procedure.

Algorithm: MATRIX-REDUCTION
INPUT: A matrix A over some field $\mathbb{F}$, and two non-negative integers r, k.
OUTPUT: A matrix having $O(r \cdot k) \times O(r \cdot k)$ entries.

1. Let C_A = COLUMN-REDUCTION(A, r, k).
2. Let R_A = COLUMN-REDUCTION(C_A^T, r, k).
3. Return R_A^T.

Figure 4.2: The dimension reduction procedure.

Algorithm: MATRIG-ALG
INPUT: A matrix A over a field $\mathbb{F}$, and two non-negative numbers r, k.
OUTPUT: Can we edit at most k entries of A to obtain a matrix of rank at most r?

1. Let A' =MATRIX-REDUCTION(A, r, k).
2. **for** each set E of k entries in A':
   (a) Replace each entry of A' indexed by an element in E by a distinct indeterminate to obtain a *mixed matrix* A'_E.
   (b) Let $\mathcal{P}$ be the set of equations obtained by setting the determinant of each $(r+1) \times (r+1)$ submatrix of A'_E to 0.
   (c) If $\mathcal{P}$ is feasible over $\mathbb{F}$ then return YESand exit.
3. Return Noand exit.

Figure 4.3: Description of the algorithm for MATRIX RIGIDITY.

Chapter 5

Rank Vertex Cover as a Natural Problem for Algebraic Compression[1]

The question of the existence of a polynomial kernelization of the Vertex Cover Above LP problem has been a longstanding, notorious open problem in Parameterized Complexity. Five years ago, the breakthrough work by Kratsch and Wahlström [66] on representative sets has finally answered this question in the affirmative. In this chapter, we present an alternative, *algebraic compression* of the Vertex Cover Above LP problem into the Rank Vertex Cover problem. Here, the input consists of a graph G, a parameter k, and a bijection between $V(G)$ and the set of columns of a representation of a matriod M, and the objective is to find a vertex cover whose rank is upper bounded by k.

5.1 Introduction

The field of Parameterized Complexity concerns the study of *parameterized problems*, where each problem instance is associated with a *parameter* k that is a non-negative integer. Given a parameterized problem of interest, which is generally computationally

[1]This chapter is based on a joint work with Fahad Panolan, Saket Saurabh and Meirav Zehavi.

hard, the first, most basic question that arises asks whether the problem at hand is *fixed-parameter tractable (FPT)*. Here, a problem Π is said to be FPT if it is solvable in time $f(k) \cdot |X|^{O(1)}$, where f is an arbitrary function that depends *only* on k and $|X|$ is the size of the input instance. In other words, the notion of FPT signifies that it is not necessary for the combinatorial explosion in the running time of an algorithm for Π to depend on the input size, but it can be confined to the parameter k. Having established that a problem is FPT, the second, most basic question that follows asks whether the problem also admits a *polynomial compression*. Here, a problem Π is said to admit a polynomial compression if there exist a problem $\widehat{\Pi}$ and a polynomial-time algorithm such that given an instance (X, k) of Π, the algorithm outputs an equivalent instance $(\widehat{X}, \widehat{k})$ of $\widehat{\Pi}$, where $|\widehat{X}| = \widehat{k}^{O(1)}$ and $\widehat{k} \leq k$. Roughly speaking, compression is a mathematical concept that aims to analyze preprocessing procedures in a formal, rigorous manner. We note that in case $\Pi = \widehat{\Pi}$, the problem is further said to admit a *polynomial kernelization*, and the output $(\widehat{X}, \widehat{k})$ is called a *kernel*.

The Vertex Cover problem is (arguably) the most well-studied problem in Parameterized Complexity [33, 27]. Given a graph H and a parameter k, this problem asks whether H admits a vertex cover of size at most k. Over the years, a notable number of algorithms have been developed for the Vertex Cover problem [15, 6, 34, 85, 24, 22, 25]. Currently, the best known algorithm solves this problem in the remarkable time $1.2738^k \cdot n^{O(1)}$ [25]. While it is not known whether the constant 1.2738 is "close" to optimal, it is known that unless the Exponential Time Hypothesis (ETH) fails, Vertex Cover cannot be solved in time $2^{o(k)} \cdot n^{O(1)}$ [58]. On the other hand, in the context of kernelization, the picture is clear in the following sense: It is known that Vertex Cover admits a kernel with $O(k^2)$ vertices and edges [15], but unless NP $\subseteq$ co-NP/poly, it does not admit a kernel with $O(k^{2-\epsilon})$ edges [30]. We remark that it is also known that Vertex Cover admits a kernel not only of size $O(k^2)$, but also with only $2k$ vertices [24, 68], and it is conjectured that this bound might be essentially tight [23].

It has become widely accepted that Vertex Cover is one of the most natural test beds for the development of new techniques and tools in Parameterized Complexity. Unfortunately, the vertex cover number of a graph is generally large—in fact, it is often linear in the size of the entire vertex set of the graph [33, 27]. Therefore, alternative parameterizations, known as *above guarantee parameterizations*, have been proposed. The two most well known such parameterizations are based on the observation that the vertex cover number of a graph H is at least as large as the *fractional vertex cover number* of H, which in turn is at least as large as the maximum size of a matching of H. Here, the fractional vertex cover number of H is the solution to the linear program that minimizes $\sum_{v \in V(H)} x_v$ subject to the constraints $x_u + x_v \geq 1$ for all $\{u, v\} \in E(H)$, and $x_v \geq 0$ for all $v \in V(H)$. Accordingly, given a graph H and a parameter k, the Vertex Cover Above MM problem asks whether H admits a vertex cover of size at most $\mu(H) + k$, where $\mu(H)$ is the maximum size of a matching of H, and the Vertex Cover Above LP problem asks whether H admits a vertex cover of size at most $\ell(H) + k$, where $\ell(H)$ is the fractional vertex cover number of H.

On the one hand, several parameterized algorithms for these two problems have been developed in the last decade [91, 89, 28, 84, 75]. Currently, the best known algorithm for Vertex Cover Above LP, which is also the best known algorithm Vertex Cover Above MM, runs in time $2.3146^k \cdot n^{O(1)}$ [75]. On the other hand, the question of the existence of polynomial kernelizations of these two problems has been a longstanding, notorious open problem in Parameterized Complexity. Five years ago, the breakthrough work by Kratsch and Wahlström [66] on representative sets has finally answered this question in the affirmative. Till date, the kernelizations given by Kratsch and Wahlström are the only known (randomized) polynomial compressions of Vertex Cover Above MM and Vertex Cover Above LP. Note that, since $\ell(H)$ is necessarily at least as large as $\mu(H)$, a polynomial compression of Vertex Cover Above LP also implies a polynomial compression of Vertex Cover Above MM. We also remark that several central problems in Parameterized Complexity, such as the Odd Cycle Transversal problem, are known to admit

parameter-preserving reductions to Vertex Cover Above LP [75]. Hence, the significance of a polynomial compression of Vertex Cover Above LP also stems from the observation that it simultaneously serves as a polynomial compression of additional well-known problems.

Recently, a higher above-guarantee parameterization of Vertex Cover, resulting in the Vertex Cover Above Lovász-Plummer, has been introduced by Garg and Philip [45]. Here, given a graph H and a parameter k, the objective is to determine whether H admits a vertex cover of size at most $(2\ell(H) - \mu(H)) + k$. Garg and Philip [45] showed that this problem is solvable in time $3^k \cdot n^{O(1)}$, and Kratsch [65] showed that it admits a (randomized) kernelization that results in a large, yet polynomial, kernel. We remark that above-guarantee parameterizations can very easily reach bars beyond which the problem at hand is no longer FPT. For example, Gutin et al. [52] showed that the parameterization of Vertex Cover above $m/\Delta(H)$, where $\Delta(H)$ is the maximum degree of a vertex in H and m is the number of edges in H, results in a problem that is not FPT (unless FPT =W[1]).

Our Results and Methods. In this chapter, we present an alternative, *algebraic compression* of the Vertex Cover Above LP problem into the Rank Vertex Cover problem. We remark that Rank Vertex Cover was originally introduced by Lovász [76] as a tool for the examination of critical graphs. Given a graph H, a parameter ℓ, and a bijection between $V(G)$ and the set of columns of a representation of a matroid M, the objective of Rank Vertex Cover is to find a vertex cover of H whose rank (see Definition 10), which is defined by the set of columns corresponding to its vertices, is upper bounded by ℓ. Note that formal definitions of the terms used in the definition of Rank Vertex Cover can be found in Section 5.3.1.

We obtain a (randomized) polynomial compression of size $\tilde{O}(k^7 + k^{4.5}\log(1/\varepsilon))$[2], where ε is the probability of failure. Here, by failure we mean that we output an instance of Rank Vertex Cover whose size is not within the claim bound or that is not equivalent

[2] $\tilde{O}$ hide factors polynomial in $\log k$

to the input instance. In the first case, we can simply discard the output instance, and return an arbitrary instance of constant size; thus, we ensure that failure only refers to the maintenance of equivalence. Our work makes use of properties of linear spaces and matroids, and also relies on elementary probability theory. One of the main challenges it overcomes is the conversion of the methods of Lovász [76] into a procedure that works over a finite field.

5.2 Preliminaries

In this chapter, the notation $\mathbb{F}$ will refer to a finite field of prime size. Accordingly, $\mathbb{F}^n$ is an n-dimensional linear space over the field $\mathbb{F}$, where a vector $v \in \mathbb{F}^n$ is a tuple of n elements from the field $\mathbb{F}$. Here, the vector v is implicitly assumed to be represented as a column vector, unless stated otherwise. The span of a subset of columns of a representable matroids is simply their linear span.

For a graph G and a vertex $v \in V(G)$, we use $G \setminus v$ to denote the graph obtained from G after deleting v and the edges incident with v.

5.2.1 Matroids

Definition 7. *A* matroid X *is a pair* $(U, \mathcal{I})$*, where* U *is a set of elements and* $\mathcal{I}$ *is a set of subsets* U*, which satisfies the following properties:* $(i)\emptyset \in \mathcal{I}$*,* (ii) *If* $I_1 \subset I_2$ *and* $I_2 \in \mathcal{I}$*, then* $I_1 \in \mathcal{I}$*, and* (iii) *If* $I_1, I_2 \in \mathcal{I}$ *and* $|I_1| < |I_2|$*, then there exists* $x \in (I_2 \setminus I_1)$ *such that* $I_1 \cup \{x\} \in \mathcal{I}$*.*

A set $I' \in \mathcal{I}$ is said to be *independent*; otherwise, it is said to be *dependent*. A set $B \in \mathcal{I}$ is a *basis* if no superset of B is independent. For example, $U_{t,n} = ([n], \{x : x \subseteq [n], |x| \leq t\})$ forms a matroid known as a *uniform matroid*. For a matroid $X = (U, \mathcal{I})$, we use $E(X), \mathcal{I}(X)$ and $\mathcal{B}(X)$ to denote the ground set U of X, the set of independent sets $\mathcal{I}$ of X,

and the set of bases of X, respectively.

We are interested in *linear matroids*, which are defined as follows. Given a matroid $X = (U, \mathcal{I})$, a matrix M having $|U|$ columns is said to *represent* X if **(i)** the columns of M are in bijection with the elements in U, and **(ii)** a set $A \subseteq U$ is independent in X if and only if the columns corresponding to A in M are linearly independent. Accordingly, a matroid is a linear matroid if it has a representation over some field. For simplicity, we use the same symbol to refer to a matroid M and its representation. For a matrix M and some subset B of columns of M, we let $M[\star, B]$ denote the submatrix of M that is obtained from M be deleting all columns outside B. The submatrix of M over a subset of rows R and a subset of columns B is denoted using $M[R, B]$.

We proceed by stating several basic definitions related to matroids that are central to our work. For this purpose, let $X = (U, \mathcal{I})$ be a matroid.

An element $x \in U$ is called a *loop* if it does not belong to any independent set of X. If X is a linear matroid, then loops correspond to zero column vectors in its representation. An element $x \in U$ is called an *isthmus* if it occurs in every basis of X. Note that for a linear matroid X, an element x is an isthmus if it is linearly independent from any subset of $U \setminus \{x\}$. For a subset $A \subseteq U$, the *rank* of A is defined as the maximum size of an independent subset of A, that is, $\text{rank}(A) := \max_{I' \subseteq A}\{|I'| : I' \in I\}$.

The *rank function* of X is the function $\text{rank} : 2^U \to \mathbb{Z}^+$ that assigns $\text{rank}(A)$ to each subset $A \subseteq U$. Note that this function satisfies the following properties.

1. $0 \leq \text{rank}(A) \leq |A|$,
2. if $A \subseteq B$, then $\text{rank}(A) \leq \text{rank}(B)$, and
3. $\text{rank}(A \cup B) + \text{rank}(A \cap B) \leq \text{rank}(A) + \text{rank}(B)$.

A subset $F \subseteq U$ is a *flat* if $\text{rank}(F \cup \{x\}) > \text{rank}(F)$ for all $x \notin F$. In words, a flat is a subset of U such that the insertion of any additional element increases its rank. Let $\mathcal{F}$ be

the set of all flats of the matroid X. For any subset A of U, the closure $\overline{A}$ is defined as $\overline{A} = \bigcap_{F \in \mathcal{F}} \{F : A \subseteq F\}$. In other words, the closure of a set is the flat of minimum rank containing it. Let F be a flat of the matroid X. A point $x \in F$ is said to be in *general position on* F if for any flat F' of X, if x is contained in $\text{span}(F' \setminus \{x\})$ then $F \subseteq F'$.

Deletion and Contraction. The deletion of an element u from X results in a matroid X', denoted by $X \setminus u$, with ground set $E(X') = E(X) \setminus \{u\}$ and set of independent sets $\mathcal{I}(X') = \{I : I \in \mathcal{I}(X), u \notin I\}$. The contraction of a non-loop element u from X results in a matroid X', denoted by X/u, with ground set $E(X') = E(X) \setminus \{u\}$ and set of independent sets $\mathcal{I}(X') = \{I \setminus \{u\} : u \in I \text{ and } I \in \mathcal{I}(X)\}$. Note that B is a basis in X/u if and only if $B \cup \{u\}$ is a basis in X. We remark that linear matroids are closed under contraction, and the representation of the contracted matroid [48] can be found in polynomial time.

When we are considering two matroids X and X/u, then for any subset $T \subseteq E(X) \setminus \{u\}$, $\overline{T}$ represents the closure of T with respect to the matroid X.

A matroid can be also be represented by a ground set and a rank function, and for our purposes, it is sometimes convenient to employ such a representation. That is, we also use a pair (U, r) to specify a matroid, where U is the ground set and r is rank function. Now, we prove several lemmas regarding operations on matroids, which are used later in the chapter.

Observation 10. *Let M be a matroid, $u \in E(M)$ be a non-loop element in M and v be an isthmus in M. Then,* $\mathsf{rank}(M/u) = \mathsf{rank}(M) - 1$ *and v is an isthmus in M/u.*

Given a matrix (or a linear matroid) A and a column $v \in A$, by moving the vector v to another vector u, we refer to the operation that replaces the column v by the column u in A.

Lemma 32. *Let $X = (U, \mathcal{I})$ be a linear matroid, $W \subseteq U$, and let $u, v \notin W$ be two elements that are each an isthmus in X. Let X' be the linear matroid obtained by moving u to a general position on the flat spanned by W. Then, v is also an isthmus in X'.*

Proof. Let u' denote the vector to which u was moved (that is in general position on the span of W). Notice that the only modification performed with respect to the vectors of X is the update of u to u'. Suppose, by way of contradiction, that v is not an isthmus in X'. Then, there exists a set of elements $S \subseteq E(X')$, where $v \notin S$, whose span contains v. If $u' \notin S$, then $S \subseteq U$, which implies that v was not an isthmus in X. Since this results in a contradiction, we have that $u' \in S$. As u' is in the span of W, v must be in the span of $(W \cup S) \setminus \{u'\}$. Since $(W \cup S) \setminus \{u'\} \subseteq U$ and $v \notin (W \cup S) \setminus \{u'\}$, we have thus reached a contradiction. □

We remark that the statement in Lemma 32 does not require the vector u to be moved to a general position on the flat, but it holds true also if u is moved to an arbitrary vector in the span of W. Even though we proved a stronger result, we stated a weaker Lemma 32 as it will be useful in its present form.

Lemma 33. *Let $X = (U, \mathcal{I})$ be a matroid and $u \in U$ be an element that is not a loop in X. If $v \in U$ is an isthmus in X, then v is also an isthmus in the contracted matroid X/u.*

Proof. Suppose, by way of contradiction, that v is not an isthmus in X/u. Then, there exists an independent set $I \in \mathcal{I}(X/u)$, where $v \notin I$, whose span contains v. In particular, this implies that $I \cup \{v\}$ is a dependent set in X/u. By the definition of contraction, $I \cup \{u\}$ is an independent set in X. As v is an isthmus in X, $I \cup \{u, v\}$ is also an independent set in X. By the definition of contraction, $I \cup \{v\}$ is an independent set in X/u, which contradicts our previous conclusion that $I \cup \{v\}$ is a dependent set in X/u. □

Lemma 34. *Let v be an element in a matroid $X = (U, \mathcal{I})$, which is not a loop in X. Let T be a subset of U such that $v \in \overline{T}$. Then,* $\mathsf{rank}_X(T) = \mathsf{rank}_{X/v}(T \setminus \{v\}) + 1$.

Proof.

$$\begin{aligned}
\mathsf{rank}_{X/v}(T \setminus \{v\}) &= \max_{I \subseteq T \setminus \{v\}} \{|I| : I \in \mathcal{I}(X/v)\} \\
&= \max_{I \subseteq T \setminus \{v\}} \{|I| : I \cup \{v\} \in \mathcal{I}(X)\} \\
&= \mathsf{rank}_X(T \cup \{v\}) - 1 \\
&= \mathsf{rank}_X(T) - 1 \quad (\text{because } v \in \overline{T}).
\end{aligned}$$

This completes the proof of the lemma. □

The lemma above can be rephrased as follows: if T is a set of elements in a matroid $X = (U, \mathcal{I})$ such that an element $v \in U$ is contained in the span of T, then the rank of T in the contracted matroid X/v is smaller by 1 than the rank of T in X.

5.3 Compression

Our objective is to give a polynomial compression of Vertex Cover Above LP. More precisely, we develop a polynomial-time randomized algorithm that given an instance of Vertex Cover Above LP with parameter k and $\varepsilon > 0$, with probability at least $1-\varepsilon$ outputs an equivalent instance of Rank Vertex Cover whose size is bounded by a polynomial in k and ϵ. It is known that there is a parameter-preserving reduction from Vertex Cover Above LP to Vertex Cover Above MM such that the parameter of the output instance is linear in the parameter of the original instance [66]. Thus, in order to give a polynomial compression of Vertex Cover Above LP to Rank Vertex Cover where the size of the output instance is bounded by $\tilde{O}(k^7 + k^{4.5}\log(1/\varepsilon))$, it is enough to give a polynomial compression of Vertex Cover Above MM to Rank Vertex Cover with the same bound on the size of the output instance. Given a graph H, we use $\mu(H)$ and $\beta(H)$ to denote the maximum size of a matching of H and the vertex cover number of H, respectively. Let (G, k) be an instance of Vertex Cover Above MM. Let $n = |V(G)|$ and I_n denote the $n \times n$

identity matrix. That is, I_n is a representation of $U_{n,n}$. Notice that (G, k) is a Yes-instance of Vertex Cover Above MM if and only if $(G, I_n, \mu(G) + k)$, with any arbitrary bijection between $V(G)$ and columns of I_n, is a Yes-instance of Rank Vertex Cover.

In summary, to give the desired polynomial compression of Vertex Cover Above LP, it is enough to give a polynomial compression of instances of the form $(G, I_n, \mu(G)+k)$ of Rank Vertex Cover where the size of the output instance is bounded by $\tilde{O}(k^7 + k^{4.5}\log(1/\varepsilon))$. Here, the parameter is k. For instances of Rank Vertex Cover, we assume that the columns of the matrix are labeled by the vertices in $V(G)$ in a manner corresponding to a bijection between the input graph and columns of the input matrix. As discussed above, we again stress that now our objective is to give a polynomial compression of an instance of the form $(G, I_n, \mu(G) + k)$ of Rank Vertex Cover to Rank Vertex Cover, which can now roughly be thought of as a polynomial kernelization. We achieve the compression in two steps.

1. In the first step, given $(G, M = I_n, \mu(G) + k)$, in polynomial time we either conclude that $(G, I_n, \mu(G) + k)$ is a Yes-instance of Rank Vertex Cover or (with high probability of success) output an equivalent instance (G_1, M_1, ℓ) of Rank Vertex Cover where the number of rows in M_1, and hence $\mathsf{rank}(M_1)$, is upper bounded by $O(k^{3/2})$. More over we also bound the bits required for each entry in the matrix to be $\tilde{O}(k^{5/2} + \log(1/\varepsilon))$ This step is explained in Section 5.3.2. Notice that after this step, the graph G_1 need not be bounded by $k^{O(1)}$.
2. In the second step, we work with the output (G_1, M_1, ℓ) of the first step, and in polynomial time we reduce the number of vertices and edges in the graph G_1 (and hence the number of rows and columns in the matrix M_1). That is, output of this step is an equivalent instance (G_2, M_2, ℓ) where the size of G_2 is bounded by $O(k^3)$. This step is explained in Section 5.3.3.

Throughout the compression algorithm, we work with Rank Vertex Cover. Notice that

the input of Rank Vertex Cover consists of a graph G, an integer ℓ, and a linear representation M of a matroid with a bijection between $V(G)$ and the set of columns of M. In the compression algorithm, we use operations that modify the graph G and the matrix M simultaneously. To employ these "simultaneous operations" conveniently, we first define (in Section 5.3.1) the notion of a *graph-matroid pair*. Then, we define deletion and contraction operations on a graph-matroid pair, and state some properties of these operations.

5.3.1 Graph-Matroid Pairs

We start with the definition of a graph-matroid pair.

Definition 8. *A pair (H, M), where H is a graph and M is a matroid over the ground set $V(H)$, is called a* graph-matroid pair.

Notice that there is natural bijection between $V(H)$ and $E(M)$, which is the identity map. Now, we define deletion and contraction operations on graph-matroid pairs.

Definition 9. *Let $P = (H, M)$ be a graph-matroid pair, and let $u \in V(H)$. The* deletion *of u from P, denoted by $P \setminus u$, results in the graph-matroid pair $(H \setminus u, M \setminus u)$. If u is not a loop in M, then the* contraction *of u in P, denoted by P/u, results in the graph-matroid pair $(H \setminus u, M/u)$. For an edge $e \in E(H)$, $P \setminus e$ represents the pair $(H \setminus e, M)$*

In this chapter, these operations are performed on graph-matroid pairs. We remark that matroid deletion and contraction can be done time polynomial in the input size. For details we refer to [48, 86].

Definition 10. *Given a graph-matroid pair $P = (H, M)$, the vertex cover number of P is defined as $\tau(P) = \min\{\text{rank}_M(S) : S \text{ is a vertex cover of } H\}$.*

For example, if M is an identity matrix (where each element is an isthmus), then $\tau(P)$ is the vertex cover number of H. Moreover, if we let M be the uniform matroid $U_{t,n}$ such

that t is at least the size of the vertex cover number of H, then $\tau(P)$ again equals the vertex cover number of H.

Let $P = (H, M)$ be a graph-matroid pair where M is a linear matroid. Recall that M is also used to refer to a given linear representation of the matroid. For the sake of clarity, we use v_M to refer explicitly to the column vector associated with a vertex $v \in V(H)$. When it is clear from context, we use v and v_M interchangeably.

Lemma 35 ([76])**.** *Let $P = (H, M)$ be a graph-matroid pair and $v \in V(H)$ such that the vector v_M is an isthmus in M, where M is a linear matroid. Let $P' = (H, M')$ be the graph-matroid pair obtained by moving v_M to a vector $v_{M'}$ in general position on a flat containing the neighbors of v, $N_H(v)$. Then, $\tau(P') = \tau(P)$.*

Proof. Note that the operation in the statement of the lemma does not change the graph H. The only change occurs in the matroid, where we map an isthmus v_M to a vector lying in the span of its neighbors. It is clear that such an operation does not increase the rank of any vertex cover. Indeed, given a vertex cover T of H, in case it excludes v, the rank of T is the same in both M and M', and otherwise, since v_M is an isthmus, the rank of T cannot increase when M is modified by replacing v_M with *any* other vector. Thus, $\tau(P') \leq \tau(P)$.

For the other inequality, let T be the set of vectors corresponding to a minimum rank vertex cover of the graph H in the graph-matroid pair P' (where we have replaced the vector v_M by the vector $v_{M'}$). In what follows, note that as we are working with linear matroids, the closure operation is the linear span. We have the following two cases:

Case 1: $v_{M'} \notin T$ In this case T is still a vertex cover of H with the same rank. Thus, $\tau(P') = \mathsf{rank}_{M'}(T) = \mathsf{rank}_M(T) \geq \tau(P)$.

Case 2: $v_{M'} \in T$ Here, we have two subcases:

- If $v_{M'} \notin \overline{T \setminus \{v_{M'}\}}$, then note that $\tau(P') = \mathrm{rank}_{M'}(T) = \mathrm{rank}_{M'}(T \setminus \{v_{M'}\}) + 1 =$

$\text{rank}_M((T \setminus \{v_{M'}\}) \cup \{v_M\}) \geq \tau(P)$. The third equality follows because v_M is an isthmus.

- If $v_{M'} \in \overline{T \setminus \{v_{M'}\}}$, then as $v_{M'}$ is in general position on the flat of its neighbors, by definition this means that all of the neighbors of $v_{M'}$ are also present in $\overline{T \setminus \{v_{M'}\}}$. Since v_M and $v_{M'}$ have the same neighbors (as the graph H has not been modified), all of the neighbors of v_M belong to in $\overline{T \setminus v_{M'}}$. Thus, $\overline{T \setminus \{v_{M'}\}}$ is a vertex cover of H. Therefore, $\tau(P') = \text{rank}_{M'}(T) = \text{rank}_{M'}(\overline{T}) = \text{rank}_{M'}(\overline{T \setminus v_{M'}}) = \text{rank}_M(\overline{T \setminus v_{M'}}) \geq \tau(P)$. The second equality crucially relies on the observation that rank of a set is equal to the rank of the span of the set.

This completes the proof of the lemma. □

Lemma 36 ([76])**.** *Let $P = (H, M)$ be a graph-matroid pair, and let v be a vertex of the graph H that is contained in a flat spanned by its neighbors. Let $P' = P/v$. Then, $\tau(P') = \tau(P) - 1$.*

Proof. Recall that the contraction of a vertex v in P results in the graph-matroid pair $P' = (H \setminus v, M/v_M)$, i.e. the vertex is deleted from the graph and contracted in the matroid. Denote the contracted matroid M/v_M by M'.

We first prove that $\tau(P) \leq \tau(P') + 1$. Let T be a minimum rank vertex cover in P', i.e. $\text{rank}_{M'}(T) = \tau(P')$. Let W be a maximum sized independent set in $\mathcal{I}(M')$ contained in T. Then, by the definition of contraction, $W \cup \{v\}$ is a maximum sized independent set in $\mathcal{I}(M)$ contained in $T \cup \{v\}$. Moreover, $T \cup \{v\}$ is a vertex cover in H, and therefore we get that $\tau(P) \leq \text{rank}_M(T \cup \{v\}) = |W \cup \{v\}| = \text{rank}'_M(T) + 1 = \tau(P') + 1$.

Now we prove that $\tau(P') \leq \tau(P) - 1$. Assume that T is a minimum rank vertex cover of P. In case $v \notin T$, it holds that all of the neighbors of v must belong T to cover edges incident to v. By our assumption, v is in the span of its neighbors in M. Therefore, v necessarily belongs to the span of T. Note that $T \setminus \{v\}$ is a vertex cover of H'. By Lemma 34, we have that $\tau(P) = \text{rank}_M(T) = \text{rank}_{M'}(T \setminus \{v\}) + 1 \geq \tau(P') + 1$. This completes the proof. □

5.3.2 Rank Reduction

In this section we explain the first step of our compression algorithm. Formally, we want to solve the following problem.

Rank Reduction
Input: An instance $(G, M = I_n, \mu(G) + k)$ of Rank Vertex Cover, where $n = \|V(G)\|$.
Output: An equivalent instance (G', M', ℓ) such that the number of rows in M' is at most $O(k^{3/2})$.

Here, we give a randomized polynomial time algorithm for Rank Reduction. More precisely, along with the input of Rank Reduction, we are given an error bound $\varepsilon > 0$, and the objective is to output a "small" equivalent instance with probability at least $1 - \varepsilon$. We start with a reduction rule that reduces the rank by 2.

Reduction Rule 31 (Vertex Deletion)**.** *Let (P, ℓ) be an instance of* Rank Vertex Cover*, where $P = (G, M)$ is a graph-matroid pair. Let $v \in V(G)$ be a vertex such that v_M is a isthmus in M. Let M_1 be the matrix obtained after moving the v_M to a vector v_{M_1} in general position on the flat spanned by $N_G(v)$. Let $P_1 = (G, M_1)$ and let $P' = P_1/v_{M_1}$. Then output $(P', \ell - 1)$*

Lemma 37. *Reduction Rule 31 is safe.*

Proof. We need to show that (P, ℓ) is a Yes-instance if and only if $(P', \ell - 1)$ is a Yes-instance, which follows from Lemmata 35 and 36. □

Lemma 38. *Let (P, ℓ) be an instance of* Rank Vertex Cover*, where $P = (G, M)$ is a graph-matroid pair. Let $(P', \ell-1)$ be the output of Reduction Rule 31, where $P' = (G', M')$. Then* $\mathsf{rank}(M') = \mathsf{rank}(M) - 2$.

Proof. In Reduction Rule 31, we move a isthmus v_M of M to a vector v_{M_1}, obtaining a matrix M_1. Note that v_{M_1} lies in the span of $N_G(v)$, and therefore v_{M_1} is not a isthmus in M_1. Hence, we have that $\mathsf{rank}(M_1) = \mathsf{rank}(M) - 1$. By the definition of general position, it holds that v_{M_1} is not a loop in M_1. Notice that $M' = M_1/v_{M_1}$. Therefore, by Observation 10, $\mathsf{rank}(M') = \mathsf{rank}(M_1) - 1 = \mathsf{rank}(M) - 2$. □

The following lemma explain how to apply Reduction Rule 31 efficiently. Later we will explain (Lemma 41) how to keep the bit length of each entries in the matrix bounded by polynomial in k.

Lemma 39. *Let M be a linear matroid with $|E(M)| = n$ and let $p > 2^n$ be an integer. Then, Reduction Rule 31 can be applied in polynomial time with success probability at least $1 - \frac{2^n}{p}$. The number of bits required for each entry in the output representation matrix is $O(\log p)$ times the number of bits required for each entry in the input representation matrix*[3].

Proof. Let F be the set of columns in M corresponding to $N_G(v)$. Using formal indeterminates $x = \{x_h : h \in F\}$, obtain a vector $g(x) = \sum_{h \in F} x_h h$. Suppose the values of the indeterminates have been fixed to some numbers x^* such that for any independent set $I \in M$ which does not span F, $I \cup \{g(x^*)\}$ is also independent. We claim that $g(x^*)$ is in general position on F. By definition, if $g(x^*)$ is not in general position, then there exists a flat F' with $g(x^*) \in \overline{F' \setminus \{g(x^*)\}}$ but it does not contain F. Let I be a basis of $F' \setminus \{g(x^*)\}$, clearly I does not span F but $I \cup \{g(x^*)\}$ is a dependent set, which is a contradiction due to the choice of x^*.

Let I be an independent set which does not span F. We need to select x in such a way that $D_{R,I}(x) = \det(M[R, I \cup \{g(x)\}])$ is not identically zero for some R. First of all, note that there is a choice of R for which the polynomial $D_{R,I}(x)$ is not identically zero and has total degree one. This is so because $D_{R,I}(x) = \sum_{h \in F} x_h \det(M[R, I \cup \{h\}])$; if it is identically

[3] We remark that we are unaware of a procedure to derandomize the application of Reduction Rule 31.

zero for every R, then $\det(M[R, I \cup \{h\}]) = 0$ which implies that every element $h \in F$ is spanned by I. Thus, this case does not arise due to the choice of I. If we choose $x \in [p]^{|F|}$ uniformly at random, for some number p, then the probability that $D_{R,I}(x) = 0$ is at most $\frac{1}{p}$ by Schwartz-Zippel Lemma. The number of independent sets in M, which does not span F, is at most 2^n. By union bound, the probability that $D_{R,I}(x) = 0$ for some I, an independent set of M, is at most $\frac{2^n}{p}$. Therefore, the success probability is at least $1 - \frac{2^n}{p}$.

The procedure runs in polynomial time and the process of matroid contraction can at most double the matrix size, this gives us the claimed bit sizes. □

In the very first step of applying Reduction Rule 31, the theorem above makes the bit sizes $O(\log p)$. On applying the the rule again the bit length of entries double each time due to Gaussian elimination performed for the step of matroid contraction. This can make the numbers very large. To circumvent this, we show that given a linear matroid $(U, \mathcal{I})$ of *low* rank and where the ground set U is *small*, along with a representation matrix M over the field $\mathbb{R}$, for a randomly chosen *small* prime q, the matrix $M \mod q$ is also a linear representation of M (see Lemma 40). To prove this result, we first observe that for any number n, the number of distinct prime factors is bounded by $O(\log n)$.

Observation 11. *There is a constant c such that number of distinct prime factors of any number n, denoted by $\omega(n)$, is at most $c \log n$.*

The well-known prime number theorem implies that

Proposition 5. *There is a constant c such that the number of distinct prime numbers smaller than or equal to n, denoted by $\pi(n)$, is at least $c\frac{n}{\log n}$.*

Lemma 40. *Let $X = (U, \mathcal{I})$ be a rank r linear matroid representable by an $r \times n$ matrix M over $\mathbb{R}$ with each entry between $-n^{c'r}(1/\delta)$ and $n^{c'r}(1/\delta)$ for some constants c' and δ. Let $\varepsilon > 0$. There is a number $c \in O(\log \frac{1}{\varepsilon})$ such that for a prime number q chosen uniformly at random from the set of prime numbers smaller than or equal to $c\frac{n^{2r+3}(n \log n + \log(1/\delta))^2}{\varepsilon}$, the matrix $M_q = M \mod q$ over $\mathbb{R}$ represents the matroid X with probability at least $1 - \frac{\varepsilon}{n}$.*

Proof. To prove that M_q is a representation of X (with high probability), it is enough to show that for any basis $B \in \mathcal{B}(X)$, the corresponding columns in M_q are linearly independent. For this purpose, consider some basis $B \in \mathcal{B}(X)$. Since B is an independent set in M, we have that the determinant of $M[\star, B]$, denoted by $\det(M[\star, B])$, is non-zero. The determinant of $M_q[\star, B]$ is equal to $\det(M[\star, B]) \mod q$. Let $a = \det(M[\star, B])$, and let $b = a \mod q$. The value b is equal to zero only if q is prime factor of b. Since the absolute value of each entry in M is at most $n^{c'r}(1/\delta)$, the absolute value of a is upper bounded by $r!n^{c'r^2}(1/\delta)^r$. By Observation 11, the number of prime factors of a is at most $c_1(\log(r!) + c'r^2 \log n + r\log(1/\delta))$ for some constant c_1. The total number of bases in X is at most n^r. Hence the cardinality of the set $F = \{z : z \text{ is a prime factor of } \det(M[\star, B]) \text{ for some } B \in \mathcal{B}(X)\}$ is at most $n^r \cdot c_1(\log(r!) + c'r^2 \log n + r\log(1/\delta)) \leq c_2 n^{r+1}(n \log n + \log(1/\delta))$ for some constant c_2.

By Proposition 5, there is a constant c_3 such that the number of prime factors less than or equal to $c\frac{n^{2r+3}(n\log n+\log(1/\delta))^2}{\varepsilon}$ is at least

$$t = c_3 c \frac{n^{2r+3}(n \log n + \log(1/\delta))^2}{\varepsilon \log(\frac{n^{2r+3}(n\log n+\log(1/\delta))^2}{\varepsilon})}.$$

The probability that M_q is not a representation of X (denote it by $M_q \not\equiv M$) is,

$$\begin{aligned} \Pr[M_q \not\equiv M] &= \Pr[q \in F] \leq \frac{|F|}{t} \\ &\leq \frac{c_2}{c_3 c} \cdot \frac{\log(\frac{n^{2r+3}(n\log n+\log(1/\delta))^2}{\varepsilon})}{n^{r+1}(n\log n + \log(1/\delta))} \cdot \frac{\varepsilon}{n} \end{aligned}$$

For any $\varepsilon > 0$, there is a number $c \in O(\log \frac{1}{\varepsilon})$ such that the above probability is at most $\frac{\varepsilon}{n}$. This completes the proof of the lemma. □

By combining Lemmata 39 and 40, we can apply Reduction Rule 31, such that each entry in the output representation matrix has bounded value.

Lemma 41. *Given $\varepsilon > 0$, Reduction Rule 31 can be applied in polynomial time with*

success probability at least $1 - \frac{\varepsilon}{n}$. *Moreover, each entry in the output representation matrix is at most* $c^{\frac{n^{2r+3}(n \log n + \log(1/\epsilon))^2}{\epsilon}}$, *where* $c \in O(\log \frac{1}{\epsilon})$.

Proof. Let M be the input representation matrix. Let $\epsilon' = \varepsilon/(2n)$. Now we apply Lemma 39 using $p > \frac{2^n}{\epsilon'}$. Let M' be the output representation matrix of Lemma 39. By Lemma 39, M' represents M with probability at least $1 - \epsilon'$. Observe that the absolute values of matrix entries are bounded by the value of q as given in Lemma 40, thus each entry in M' has absolute value bounded by $n^{c'n}/\epsilon^2$ for some constant c'. Now, applying Lemma 40 again completes the proof. □

We would like to apply Reduction Rule 31 as may times as possible in order to obtain a "good" bound on the rank of the matroid. However, for this purpose, after applying Reduction Rule 31 with respect to some isthmus of the matroid, some other isthmuses need to remain isthmuses. Thus, instead of applying Reduction Rule 31 blindly, we choose vectors v_M whose vertices belong to a predetermined independent set. To understand the advantage behind a more careful choice of the vectors v_M, suppose that we are given an independent set U in the graph G such that every vertex in it is a isthmus in the matroid. Then, after we apply Reduction Rule 31 with one of the vertices in U, it holds that every other vertex in U is still a isthmus (by Lemma 32 and Observation 10). In order to find a large independent set (in order to apply Reduction Rule 31 many times), we use the following two known algorithmic results about Vertex Cover Above MM.

Lemma 42 ([75])**.** *There is a* $2.3146^k \cdot n^{O(1)}$*-time deterministic algorithm for* Vertex Cover Above MM.

Recall that for a graph G, we let $\beta(G)$ denote the vertex cover number of G.

Lemma 43 ([82])**.** *For any* $\epsilon > 0$*, there is a randomized polynomial-time approximation algorithm that given a graph* G*, outputs a vertex cover of* G *of cardinality at most* $\mu(G) + O(\sqrt{\log n})(\beta(G) - \mu(G))$*, with probability at least* $1 - \epsilon$.

In what follows, we also need the following general lemma about linear matroids.

Lemma 44 ([48])**.** *Let M be an $a \times b$ matrix representing some matroid. If M' is a matrix consisting of a row basis of M then M' represents the same matroid as M.*

We are now ready to give the main lemma of this subsection.

Lemma 45. *There is a polynomial time randomized algorithm that given an instance $(G, M = I_n, \mu(G) + k)$ of* Rank Vertex Cover *and $\hat{\varepsilon} > 0$, with probability at least $1 - \hat{\varepsilon}$ outputs an equivalent instance (G', M', ℓ) of* Rank Vertex Cover *such that the number of rows in M' is at most $\mathcal{O}(k^{3/2})$. Here, M' is a matrix over the field $\mathbb{R}$ where each entry is $\tilde{\mathcal{O}}(k^{5/2} + \log(1/\hat{\varepsilon}))$ bits long.*

Proof. Recall that $n = |V(G)|$. If $k \leq \log n$, then we use Lemma 42 to solve the problem in polynomial time. Next, we assume that $\log n < k$. Let $\delta = \hat{\varepsilon}/2$. Now, by using Lemma 43, in polynomial time we obtain a vertex cover Y of G of size at most $\mu(G) + c' \sqrt{\log n} \cdot k \leq \mu(G) + c' \cdot k^{3/2}$, with probability at least $1 - \delta$, where c' is some constant. If we fail to compute such a vertex cover, then we output an arbitrary constant sized instance as output; and this will happen only with probability at most δ. Otherwise, let $S = V(G) \setminus Y$. Since Y is a vertex cover of G, we have that S is an independent set of G. Hence, $|S| \geq n - (\mu(G) + c' \cdot k^{3/2})$. Since $M = I_n$, all the elements of M, including the ones in S, are isthmuses in M. Now, we apply Reduction Rule 31 with the elements of S (one by one). By Lemma 32 and Observation 10, after each application of Reduction Rule 31, the remaining elements in S are still isthmuses. In particular, we apply Reduction Rule 31 $|S|$ many times. Let (G', M', ℓ) be the instance obtained after these $|S|$ applications of Reduction Rule 31 using Lemma 41 (substituting $\varepsilon = \delta$ in Lemma 41).

By Lemma 38, we know that after each application of Reduction Rule 31, the rank reduces

by 2. Hence,

$$\begin{aligned}
\mathsf{rank}(M') &= \mathsf{rank}(M) - 2|S| \\
&= n - 2\left(n - (\mu(G) + c' \cdot k^{3/2})\right) \\
&= -n + 2\mu(G) + 2c' \cdot k^{3/2} \leq 2c' \cdot k^{3/2} \qquad \text{(because } 2\mu(G) \leq n\text{)}.
\end{aligned}$$

During each application of Reduction Rule 31, by Lemma 44, we can assume that the number of rows in the representation matrix is exactly same as the rank of the matrix. Now, we return (G', M', ℓ) as the output. Notice that the number of rows in M' is at most $O(k^{3/2})$.

Now, we analyze the probability of success. As finding the approximate vertex cover Y using Lemma 43 fails with probability at most $\delta = \frac{\hat{\varepsilon}}{2}$, in order to get the required success probability of $1-\hat{\varepsilon}$, $|S|$ applications of Reduction Rule 31 should succeed with probability at least $1 - \frac{\hat{\varepsilon}}{2}$. We suppose that the matrix $M = I_n$ is over the field $\mathbb{R}$. Recall that the instance (G', M', ℓ) is obtained after $|S|$ applications of Reduction Rule 31. The failure probability of each application of Reduction Rule 31 is at most $\frac{\delta}{n}$. Hence, by union bound the probability failure in at least one application of Reduction Rule 31 is at most δ. Hence the total probability of success is at least $1 - (\delta + \delta) = 1 - \hat{\varepsilon}$. By Lemma 41 each entry in the output representation matrix is at most $c\frac{n^{2r+3}(n \log n + \log(2/\hat{\varepsilon}))^2}{\hat{\varepsilon}}$. Hence the bits required to represent an entry in M' is at most $\tilde{O}(r \log n + \log(2/\hat{\varepsilon})) = \tilde{O}(k^{5/2} + \log(1/\hat{\varepsilon}))$. □

5.3.3 Graph Reduction

In the previous subsection, we have seen how to reduce the number of rows in the matroid. In this subsection, we move to second step of our compression algorithm. That is, to reduce the size of the graph. Formally, we want to solve the following problem.

Graph Reduction
Input: An instance (G', M, ℓ) of Rank Vertex Cover such that the number of rows in M is at most $O(k^{\frac{3}{2}})$
Output: An equivalent instance (G'', M', ℓ) such that $\lvert V(G'')\rvert, \lvert E(G'')\rvert \leq O(k^3)$

Here, we give an algorithm to reduce the number of edges in the graph. Having reduced the number of edges, we also obtain the desired bound on the number of vertices (as isolated vertices are discarded). Towards this, we first give some definitions and notations. In this section, we use $\mathbb{F}$ to denote either a finite field or $\mathbb{R}$.

Definition 11 (Symmetric Square)**.** *For a set of vectors S over a field $\mathbb{F}$, the* symmetric square*, denoted by $S^{(2)}$, is defined as $S^{(2)} = \{uv^T + vu^T : u, v \in S\}$, where the operation is matrix multiplication. The elements of $S^{(2)}$ are matrices. We can define the rank function $r^{(2)} : S^{(2)} \to \mathbb{Z}$ by treating the matrices as "long" vectors over the field $\mathbb{F}$.*

With a rank function $r^{(2)}$, the pair $(S^{(2)}), r^{(2)})$ forms a matroid. For details we refer the reader to [76].

Definition 12 (2-Tuples Meeting a Flat)**.** *For a flat F in a linear matroid S (here S is a set of vectors), the* set of 2-tuples meeting *F is defined as*

$$F_2 := \{uv^T + vu^T \ : \ v \in F, u \in S\}.$$

The dot product of two column vectors $a, b \in \mathbb{F}^n$ is the scalar $a^T b$ and is denoted by $\langle a, b\rangle$. Two properties of dot product are $(i)\langle a, b\rangle = \langle b, a\rangle$ and $(ii)\langle a, b + c\rangle = \langle a, b\rangle + \langle a, c\rangle$.

Definition 13. *Given a vector space $\mathbb{F}^d$ and a subspace F of $\mathbb{F}^d$, the* orthogonal space *of F is defined as $F^\perp = \{x \in \mathbb{F}^d \ : \ \langle y, x\rangle = 0$ for all $y \in F\}$.*

The following observation can be proved using associativity of matrix multiplication and dot product of vectors.

Observation 12. *Let u, v, w be three n-length vectors. Then, $uv^T w = \langle v, w \rangle u$.*

For the sake of completeness, we prove the following lemmas using elementary techniques from linear algebra.

Lemma 46 ([76])**.** *For any flat F in a linear matroid S with rank function r, it holds that F_2 (the set of 2-tuples meeting F) forms a flat in the matroid $(S^{(2)}, r^{(2)})$.*

Proof. Suppose, by way of contradiction, that F_2 is not a flat. Then, there exist $a, b \in S$ such that $e = ab^T + ba^T \in S^{(2)}$ is not in F_2 and

$$r^{(2)}(F_2 \cup \{e\}) = r^{(2)}(F_2).$$

As e lies in the span of F_2, there exist scalars λ_{uv} such that

$$ab^T + ba^T = \sum_{u \in F, v \in S} \lambda_{uv}(uv^T + vu^T). \tag{5.1}$$

Note that neither a nor b belongs to F, because if at least one of them belongs to F, then e lies in F_2 (by the definition of F_2). Therefore, $F \neq S$ and it is a proper subspace of S, which implies that $F^\perp$ is non-empty (follows from Proposition 13.2 in [61]). Pick an element $x \in F^\perp$. By right multiplying the column matrix x with the terms in Equation 5.1, we get

$$\begin{aligned} ab^T x + ba^T x &= \sum_{u \in F, v \in S} \lambda_{uv}(uv^T x + vu^T x) \\ \langle b, x \rangle a + \langle a, x \rangle b &= \sum_{u \in F, v \in S} \lambda_{uv} \langle v, x \rangle u + \langle u, x \rangle v \\ &= \sum_{u \in F, v \in S} \lambda_{uv} \langle v, x \rangle u \end{aligned} \tag{5.2}$$

The second equality follows from Observation 12, and the third equality follows from the fact that $\langle u, x\rangle = 0$ (because $u \in F$ and $x \in F^{\perp}$). Now, by taking dot product with x, from Equation 5.2, we have that

$$2\langle a, x\rangle\langle b, x\rangle = \sum_{u\in F, v\in S} \lambda_{uv}\langle v, x\rangle\langle u, x\rangle = 0 \tag{5.3}$$

The last equality follows from the fact that $\langle u, x\rangle = 0$. As the choice of x was arbitrary, Equations 5.2 and 5.3 hold for all $x \in F^{\perp}$.

By Equation 5.3, for all $x \in F^{\perp}$, at least one of $\langle b, x\rangle$ or $\langle a, x\rangle$ is zero. If exactly one of $\langle b, x\rangle$ or $\langle a, x\rangle$ is zero for some $x \in F^{\perp}$, then at least one of a or b is a linear combination of vectors from F (by Equation 5.2) and hence it belongs to F, which is a contradiction (recall that we have argued that both a and b do not belong to the flat F). Now, consider the case where both $\langle b, x\rangle$ and $\langle a, x\rangle$ are zero for all $x \in F^{\perp}$. Then, both a and b belong to $F^{\perp\perp}$. Since $F^{\perp\perp} = F$ (in the case F is over a finite field, see Theorem 7.5 in [54]), again we have reached a contradiction. □

For a graph-matroid pair $P = (H, M)$ (here M represents a set of vectors), define $\mathcal{E}(P) \subseteq M^{(2)}$ as

$$\mathcal{E}(P) = \{uv^T + vu^T : \{u, v\} \in E(H)\}.$$

Note that $\mathcal{E}(P)$ forms a matroid with the same rank function as the one of $M^{(2)}$. Moreover, the elements of $\mathcal{E}(P)$ are in correspondence with the edges of H. For simplicity, we refer to an element of $\mathcal{E}(P)$ as an edge. Using Lemma 46, we prove the following lemma.

Lemma 47 ([76])**.** *Let $P = (H, M)$ be a graph-matroid pair, and let $r^{(2)}$ be the rank function of $\mathcal{E}(P)$. For an edge e that is not an isthmus in $(\mathcal{E}(P), r^{(2)})$, it holds that $\tau(P \setminus e) = \tau(P)$.*

Proof. The deletion of edges cannot increase the vertex cover number, thus $\tau(P \setminus e) \leq$

$\tau(P)$. Next, we show that it also holds that $\tau(P \setminus e) \geq \tau(P)$.

Let T be a vertex cover of $H \setminus e$. Notice that $\overline{T}$ is a flat in M. Denote $e = \{u, v\}$ and $F = \overline{T}$. If at least one of u or v lies in F, then F is a vertex cover of H and hence $\tau(P \setminus e) \geq \tau(P)$. Hence, to conclude the proof, it is sufficient to show that at least one of u or v lies in F. Suppose, by way of contradiction, that $u, v \notin F$. Then, the edge $e = uv^T + vu^T$ does not belong to F_2 (the set of 2-tuples meeting F). By Theorem 47, we have that F_2 is a flat in $(M^{(2)}, r^{(2)})$. Since F is a vertex cover of $H \setminus e$, by the definition of F_2 and $\mathcal{E}(P)$, we have that $\mathcal{E}(P) \setminus \{e\} \subseteq F_2$. Recall that e is not an isthmus in $(\mathcal{E}(P), r^{(2)})$. This implies that e belongs to the closure of $\mathcal{E}(P) \setminus \{e\}$, and hence it belongs to its superset F_2. We have thus reached a contradiction. This completes the proof. □

Using Lemma 47, we get the following bound on the number of edges.

Lemma 48. *Let (H, M, ℓ) be an instance of* Rank Vertex Cover *and $r = \mathsf{rank}(M)$. Applying the reduction given by Lemma 47 on (H, M) exhaustively results in a graph with at most $\binom{r+1}{2}$ edges.*

Proof. Observe that the dimension of the matroid $(\mathcal{E}(P), r^{(2)})$ is bounded by $\binom{r+1}{2}$, and the reduction given by Lemma 47 deletes any edge that is not an isthmus in this matroid. In other words, once the reduction can no longer be applied, every edge is an isthmus in the matroid $(\mathcal{E}(P), r^{(2)})$, and hence the graph has at most $\binom{r+1}{2}$ edges. □

Lemma 48 bounds the number of edges of the graph. To bound the number of vertices in the graph, we apply the following simple reduction rule.

Reduction Rule 32. *Let (G, M, ℓ) is an instance of* Rank Vertex Cover. *For $v \in V(G)$ of degree 0 in G, output $(G \setminus v, M \setminus v, \ell)$.*

Reduction Rule 32 and Lemma 48 lead us to the main result of this subsection:

Theorem 33. *There is a polynomial time algorithm, which given an instance* (G', M', ℓ) *of* Rank Vertex Cover *such that the number of rows in* M *is at most* $O(k^{\frac{3}{2}})$*, outputs an equivalent instance* (G'', M'', ℓ) *such that* $|V(G'')|, |E(G'')| = O(k^3)$*. Here,* M'' *is a restriction of* M'*.*

By combining both the steps we get a polynomial compression of size $\tilde{O}(k^7 + k^{4.5} \log(1/\varepsilon))$ for Vertex Cover Above LP.

Chapter 6

Parameterized Complexity of Strip Packing and Minimum Volume Packing[1]

We study the parameterized complexity of Minimum Volume Packing and Strip Packing. In the two dimensional version, the input consists of a set of rectangles S with integer side lengths. In the Minimum Volume Packing problem, given a set of rectangles S and a number k, the goal is to decide if the rectangles can be packed in a bounding box of volume at most k. In the Strip Packing problem we are given a set of rectangles S, numbers W and k; the objective is to find if all the rectangles can be packed in a box of dimensions $W \times k$. We prove that the 2-dimensional Volume Packing is in FPT by giving an algorithm that runs in $(2 \cdot \sqrt{2})^k \cdot k^{O(1)}$ time. We also show that Strip Packing is W[1]-hard even in two dimensions and give an FPT algorithm for a special case of Strip Packing. Some of our results hold for the problems defined in higher dimensions as well.

[1]This chapter is based on a joint work with Pradeesha Ashok, Sudeshna Kolay and Saket Saurabh [5].

6.1 Introduction

The problem of packing objects optimally inside a bin/box is studied in various forms. Two prominent examples are the Bin Packing problem and the Knapsack problem. We study generalizations of these problems in a geometric setting. A natural generalization of the Knapsack problem to two dimensions is the Cutting Stock problem [8]. More specifically, we study the problem of packing axis-parallel rectangles inside a rectangular box when only translations are allowed. This restriction of not allowing rotations does not make the problem artificial, as it still finds practical application [71]. These problems also find applications in VLSI design [83], scheduling [100], packing television commercial into station breaks [13] etc.

We consider two versions of this problem. In the Strip Packing problem, given a set of n axis-parallel rectangles and a rectangular box (strip) of width W, the goal is to pack all rectangles into this strip so that the height used is minimized. In the Minimum Volume Packing problem, given a set of n axis-parallel rectangles, goal is to pack these rectangles in a rectangular container so that the volume of the container is minimized. We also study these problems in higher dimensions. Now we formally define the problems:

d-Strip Packing

Input: A list of boxes $\{b_i \in \mathbb{N}^d \mid 1 \leq i \leq n\}$ and a vector of positive integers $W \in \mathbb{N}^{d-1}$.

Output: The minimum integer k such that all the boxes can be packed under translation into a strip with dimension vector $W \times k$, k being the height of the strip.

Volume packing in two-dimensions is an active area of research due to its immense practical importance [56].

d-Volume Packing

Input: A list of n boxes $\{b_i \in \mathbb{N}^d \mid 1 \leq i \leq n\}$.

Output: The minimum volume rectangular container into which the input boxes can be packed under translation.

The decision versions of both these problems are proved to be NP-Complete and are well studied in the context of approximation algorithms. Two-dimensional Strip Packing admits an AFPTAS [63]. Since Bin Packing, which is a special case of Strip Packing, cannot have a PTAS [4], the same holds for Strip Packing. For recent approximation algorithms and results on Minimum Volume Packing see [3]. The online version of these problems are also studied [53, 102, 59]. For a survey of these problems, we refer the reader to [71, 26].

We consider the standard parameterized version of these problems, namely the case when the parameter is the size of the solution. To the best of our knowledge, this is the first study of the parameterized versions of these problems. For the Strip Packing problem, we prove that the parameterized version is W-hard for the general case. However, the problem becomes FPT for a special case where the dimensions of the boxes and therefore the number of types of boxes is bounded by a constant, this special case is also inspired by a variant of the Bin Packing problem with similar constraints [47]. We also consider several special kind of input box dimensions where PTAS and even polynomial time algorithms exist.

Our results:

1. We prove that 2-Minimum Volume Packing is in FPT by giving an algorithm with running time $(2 \cdot \sqrt{2})^k \cdot k^{O(1)}$. This algorithm can be generalized to d-dimensions.

2. We prove that Strip Packing is W-hard even in two dimensions.

3. We consider a special case of Strip Packing where every dimension of the input boxes is bounded by a constant. For this case, we show that Strip Packing problem admits an FPT algorithm.

4. We consider some special cases of Strip Packing where the input rectangles are squares whose dimensions come from a special set and show that they are solvable

in polynomial time.

Section 6.2 gives some notations and definitions that will be used in subsequent sections. In Section 6.3, we discuss the FPT algorithm for Minimum Volume Packing problem. Section 6.4 gives algorithmic and hardness results for Minimum Strip Packing. Section 6.5 gives polynomial time algorithms for special cases.

6.2 Preliminaries

For the context of this chapter, it is enough to know that Bin Packing, in unary representation and parameterized by the number k of bins allowed, is W[1]-hard [60]. To show W[1]-hardness of a given problem, it is enough to give an FPT-reduction from Bin Packing, in unary representation and parameterized by k, to the given problem.

Packing: In this chapter, we only consider axis parallel boxes and containers with dimensions that are positive integer values. When the boxes are in $\mathbb{N}^2$ they are referred to as either *integral* rectangles or rectangles. The faces of rectangles are also referred to as edges. The dimensions of a box in $\mathbb{N}^d$ is denoted by a vector of length d, where the i^{th} index indicates the length of the projection of the axis-parallel box onto the i^{th} axis in $\mathbb{R}^d$. Similarly, we can define a dimension vector for a container. For a dimension d, given a set of positive integers $\ell_1, \ldots, \ell_d \in \mathbb{N}$, a container C in d-dimensions with dimension vector $(\ell_1, \ldots, \ell_d)$ is formally defined to be the set of points $R = \{(x_1, \ldots, x_d) \in \mathbb{R}^d | x_i \in [0, \ell_i], \forall i \in [d]\}$. An i^{th} lower *face* of R is the set $R \cap \{x_i = 0\}$. Similarly an i^{th} upper *face* of R is the set $R \cap \{x_i = \ell_i\}$. Each rectangle in d dimensions has $2 \cdot d$ faces in total. A *corner* of a rectangle is defined as the point of intersection of d non-parallel faces. The *corner* of a rectangle which is at closest distance (L_2-norm) to the origin shall be referred to as a *min-corner* point. A dimension vector, where each index stores a positive integer, is also referred to as an *integral* dimension vector. Given a dimension vector $(\ell_1, \ldots, \ell_d)$, the *volume* corresponding to the dimension vector is $\prod_{i=1}^{d} \ell_i$. Two boxes with the same dimension vector

will be referred to as the same *type*. In this chapter, we allow the boxes to be packed in such a way that they are axis-parallel and only translations are allowed.

For ease of presentation, in the case of $\mathbb{R}^2$, we refer to directions like up, left, down, right. In 2-dimension, the lower left corner of the container is assumed to be at the origin.

Given a container and input boxes, such that all dimension vectors are integral, we call an arrangement of rectangles inside a container a *packing* if the rectangles are completely contained inside the container and do not overlap (except at the faces). A *packing* is said to be *integral* if the input rectangles are packed in such a way that each corner point of every rectangle is placed on a point with integer coordinates. A packing is called as a *perfect packing* if for each box b and each axis i, the minimum value taken by the i^{th} coordinate, over all points belonging to a face of b, cannot be decreased, while keeping the coordinates of all other boxes fixed and maintaining a packing of all input boxes. Given a fixed container C and a collection of boxes $\mathcal{R}$, a packing of $\mathcal{R}$ in C is said to be a *tiling* if every point inside C is contained inside some box in $\mathcal{R}$. For this to happen the sum of *volume* of boxes must be equal to the *volume* of the container.

Parameterized Packing Problems: In this chapter, we look at the following parameterized variants of d-Strip Packing and d-Volume Packing.

d-p-Strip Packing	**Parameter:** k
Input: A list of boxes $\{b_i \in \mathbb{N}^d \mid 1 \leq i \leq n\}$, $W \in \mathbb{N}^{d-1}$ and $k \in \mathbb{N}$.	
Question: Is there a container with dimensions $W \times k$ such that all the boxes can be packed into it under axis-parallel translation?	

d-p-Strip Packing with bounded dimensions	**Parameter:** k
Input: A list of boxes $\{b_i \in \mathbb{N}^d \mid 1 \leq i \leq n\}$, where each side length of a box is at most $\ell \in \mathbb{N}$, a vector of positive integers $W \in \mathbb{N}^{d-1}$ and $k \in \mathbb{N}$.	
Question: Is there a container with dimensions $W \times k$ such that all the boxes can be packed under axis-parallel translation into it?	

d-p-Volume Packing	**Parameter:** k
Input: A list of boxes $\{b_i \in \mathbb{N}^d \mid 1 \leq i \leq n\}$ and a positive integer k.	
Question: Is there a container with volume at most k such that all the rectangles can be packed under translation into the container?	

6.3 Volume Packing

In this section, we show that d-p-Volume Packing is FPT. In order to prove this, we first derive a few relations between optimal packings and integral packings. These relations will be useful in later sections as well.

Lemma 49. *Keeping the dimension vector of the container same, any packing of boxes can be changed into a* perfect packing. *Moreover, every* perfect packing *of boxes with integral dimension vectors is also an* integral packing.

Proof. We apply a sweeping line algorithm to rearrange the rectangles in 2-dimensions. In higher dimensions, it may be looked upon as a sweeping hyperplane algorithm. We call a *hyperplane* perpendicular to the i^{th} axis in d dimensions as an i-plane. For each $i \in [d]$, we take an i-plane and sweep it towards the positive direction of the i^{th} axis starting from the zero value of the i^{th} axis. The moment when the face of a box gets completely contained inside an i-plane is referred to as an *event*. At every *event* t, we keep a list L_t^i of boxes where a *face*, which is perpendicular to the i^{th} axis, is contained in it completely. As the packing is axis parallel there are only two faces for every box

which will result in an *event* and they can be unambiguously identified as an upper face or a lower face depending on the value of their i^{th} coordinate. Divide L_t^i into two disjoint sets depending on which face of the box is completely contained in it; $L_{t,u}^i$ denotes the boxes where the upper face lies on the sweeping hyperplane at t, and $L_{t,l}^i$ denotes the boxes with the lower face lying on the sweeping hyperplane at t. Let $F_t^i \subseteq L_{t,l}^i$ be the set of boxes whose lower face does not intersect with the upper face of any box in $L_{t,u}^i$. Observe that the boxes in F_t^i can be "slid down" parallel to the i^{th} axis until either they hit a face of a rectangle below or hit a face of the container. More formally, it is possible to change the position of a box in F_t^i, without changing the volume of the bounding container, such that the value at the i^{th} coordinate of each point in the box is strictly less than the current value. For each box in F_t^i, we change the packing, such that all points in the boxes of F_t^i have the minimum possible value at the i^{th} coordinate, without changing the position of boxes not in F_t^i. This operation can be thought of as "moving down" of boxes in F_t^i. After all the boxes in F_t^i have been moved down, any box in $L_{t,l}^i \setminus F_t^i$ intersects with some rectangle face in $L_{t,u}^i$. The invariant maintained is that the boxes whose lower face has been processed by an *event* have their lower face intersecting with the upper face of some rectangle whose upper face has been processed by an event. As the boxes have integral dimension vectors, inductively it gives us the result that, when the i-plane has swept through the lower face of every box in the packing, the faces perpendicular to the i^{th} axis have integral co-ordinates. We keep repeating this procedure for every $i \in [d]$ until none of the i-plane sweep results in the set F_t^i being non-empty. It is possible that multiple sweeps have to be made for each i-axis. At the end of the procedure the corner points of each box get integral co-ordinates as they are the intersection points of the faces of the boxes.

To see why this procedure terminates, consider the sum S_b of the coordinates of the *min-corner* point of each box b in the packing, let $S = \sum_{i=1}^{b} S_b$ be the sum of S_bs. After each i-plane has been swept once, by the invariant proved above, each corner point has integral coordinates. As the container is in the positive quadrant, S is an integer greater than or equal to zero, now if any i-sweep results in F_t^i being non-empty, the value of S drops by

at least one after the sweep. Since S is bounded from below by zero and is a finite number at the start of the algorithm, the algorithm has to terminate.

With this proof, we have also proved that an arbitrary *packing* can be converted into an *integral packing* for boxes with integral dimension vectors. Thus, given any packing, we can constructively convert it into an integral perfect packing without using extra volume. For the purpose of our proofs it is enough to know that an integral perfect packing exists that is also optimal. □

As an easy consequence of the above we have the following Lemma.

Lemma 50. *There is an integral perfect packing for the input boxes, such that the volume of the container in which this packing is done is minimized.*

Now we are ready to design an FPT algorithm for d-p-Volume Packing. To illustrate the idea, we first show that 2-p-Volume Packing is FPT.

Theorem 34. 2-p-Volume Packing *is FPT with running time* $(2 \cdot \sqrt{2})^k \cdot k^{O(1)}$.

Proof. Let the input set of rectangles be represented using the set $B = \{b_i = (w_i, h_i) | i \in [n]\}$, with integers w_i and h_i being the width and height of the rectangle b_i respectively. Each rectangle contributes an area of at least one. Therefore, if k is less than the number of rectangles we can safely output NO. Thus, in the rest of the proof, we assume that $n \leq k$.

Fixing a container: First, we enumerate the dimensions of a rectangle having area k. For each choice, say $k_1 \times k_2$, of the dimensions of a potential container C, we do the following. We assume that C is kept on the co-ordinate plane with lower-left corner at the origin, each point having integral co-ordinates inside C is referred to as a grid point. We will construct a set of *tilings* of the container C using rectangles. For each tiling, we try to fit each rectangle of B inside some rectangle of the tiling. The key observation is that that any integral perfect packing of C using B can be extended into a tiling by introducing

some extra 1×1 rectangles in B used to fill in the unoccupied area in a packing. By Lemma 49, we can assume that there is an optimal packing that is integral and perfect. Also, Lemma 50 implies that a container with optimal area will be such that k_1 and k_2 are integers. Thus, there are at most k choices of containers, for which we enumerate a set of integral tilings and try to derive an optimal integral perfect packing from at least one of the tilings.

Grid Points in a Tiling: A grid point is referred to as an interior point if it is not contained on any edge of C. For any tiling of a rectangular container, there are a total of eight ways in which the edges of the participating rectangles, or tiles, may pass through an interior grid point of a container in a tiling. We refer to each configuration of edges at a grid point as a *meeting*. Figure 6.1 illustrates the *meetings* for an interior grid point. The *meeting* 6.1(a) corresponds to a corner point of four tiling rectangles. The *meeting* 6.1(b) corresponds to a grid point contained on the edges of two adjacent rectangular *tiles*. The *meeting* 6.1(c) corresponds to a grid point which is the corner point of two *tiles* and contained on the edge of another tiling rectangle. The *meeting* 6.1(d) corresponds to a grid point contained inside a tiling rectangle. The *meeting* corresponding to a corner point of a container consists of two unit length solid lines intersecting at 90°. There are exactly two *meetings* which correspond to a non-corner edge point of a container. The part of a *meeting* on the container edge must be solid, while the unit segment perpendicular to it at the centre can be either dotted or solid.

For any *meeting* M, the grid point will be denoted as M_p and will be called a *meeting point*. There are exactly four line segments incident to a meeting point. The lines incident on M_p are of two types, dotted or solid. The solid line segments correspond to the edges of the tiles that have M_p as a corner point. The dotted line segments denotes tiles where M_p is not a corner point. There can be at most 4 tiling rectangles that can intersect at M_p.

Generating a Configuration: We denote the dimensions of the container by $k_1 \times k_2$ for a rectangular grid R_C having an area of k units. The grid R_C is assumed to be contained

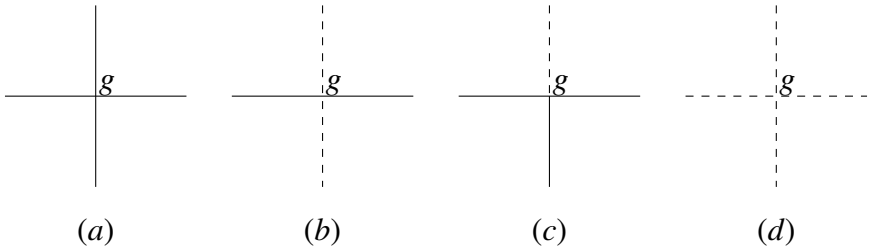

Figure 6.1: The 8 possible meetings of a two-dimensional interior grid point g: (a) All solid lines, (b) There is one other meeting obtained by rotation through 180°, (c) There are three other meetings obtained by rotation through multiples of 90° and (d) All dotted lines.

in the co-ordinate plane with lower left corner at the origin. Each point in it is in natural correspondence with the grid points of the actual container. For any two points u and v in R_C at a distance of one unit from each other, the axis-parallel line segment uv of unit length is referred to an an edge. We say that two points u, v in R_C are adjacent if they are connected by an edge. Note that the line segment uv does not contain any other grid point. A cell in R_C is defined as a minimal square containing exactly four grid points at its corners. By minimality, there is no grid point contained inside a cell of R_C. Two cells in R_C are said to be adjacent if they share an edge.

A configuration of R_C is obtained by associating a solid or a dotted line with each edge contained in it. Consider the set of at most $(2 \cdot \sqrt{2})^k$ configurations constructed as follows. Firstly, we construct a new grid, referred to as an *m-grid* on top of the old grid points in R_C as follows. Draw a line passing through the point $(1, 0)$ at 45° to the x-axis in the first quadrant and draw lines parallel to it spaced regularly at a distance of $\sqrt{2}$ from each other. We call the family of lines as $\mathcal{L}$. Similarly, draw a line passing through the point $(1, 0)$, but now at 135° to the x-axis. Then draw lines parallel to it spaced regularly at a distance of $\sqrt{2}$ from each other. We call this family of lines $\mathcal{L}_\perp$. The *m-grid* consists of the lines of $\mathcal{L}$ and $\mathcal{L}_\perp$ restricted to the grid R_C. The intersection points of $\mathcal{L}$ and $\mathcal{L}_\perp$ define the grid points of *m-grid*. We analogously refer to a cell in the *m-grid* as an *m-cell*, it has an area of 2 units. For clarity, it is depicted in Figure 6.2. Associate a meeting with each of the internal grid point in R_C not contained on the *m-grid* i.e. with the *m-cells* contained

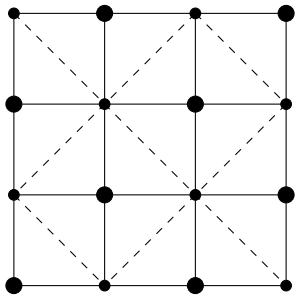

Figure 6.2: An R_C grid and its m-grid, the lower left point is the origin. The solid lines are grid lines. The dashed lines represent the m-grid lines. The large black vertices represent the internal grid points of each m-cell.

completely inside R_C. Observe that this associates a solid or a dotted line with each of the edges in R_C, except some edges contained in half-*m-cells*, with centres lying on the boundary of R_C. In a tiling configuration, the boundary of R_C is always selected i.e. it is solid. So we only need to make a choice for one line in each half-*m-cell*, with its centre at the boundary. Take two half-*m-cells* and observe that there are at most 4 choices for the undetermined edges in them. Finally, the quarter-m-cells can occur only at the corner of R_C and both their edges are determined (solid). The number of *m-cells* in R_C is $\lfloor \frac{k}{2} \rfloor$, thus the total number of configurations is at most $8^{\frac{k}{2}}$ which is $(2 \cdot \sqrt{2})^k$.

Obtaining a Tiling: Let $\mathcal{R}_C$ be a configuration. We call two adjacent cells in $\mathcal{R}_C$ *connected* if they both share a dotted edge. We construct an undirected graph G_C with the cells in $\mathcal{R}_C$ being its vertex set. For any two vertices $u, v \in V(G_C)$, (u, v) is an edge in G_C if the cells corresponding to u and v are *connected*. For each connected component in G_C, we check if the cells corresponding to the connected component in G_C form a rectangle in R_C. This can be done in polynomial time. If all the connected components in G_C correspond to a rectangle in R_C then we have generated a *tiling* of C. For a tiling $\mathcal{R}_C$, the connected components $C_1, C_2, \ldots, C_m$ of G_C correspond to the set of tiles $T = \{t_1, t_2, \ldots, t_m\}$. Note that t_i and t_j may have the same dimensions for $i \neq j$.

Obtaining a Packing: We say a rectangle a is *compatible* with b if the dimension vector of a and b are the same. Informally, it means that the rectangle a can be translated to exactly cover b. Next, for a tiling $\mathcal{R}_C$, we define an undirected bipartite graph M_C as

follows. The graph M_C consists of vertex sets $B \uplus T$, where B corresponding to the input collection of rectangles and T corresponds to the set of tiles in $\mathcal{R}_C$. For $b \in B$ and $t \in T$ there is an edge in M_C if b is *compatible* with t. Finally, a tiling T of $\mathcal{R}_C$ contains a packing of B if and only if there exists a matching in M_C which saturates B.

Proof of Correctness: By Lemma 50, there is a container with optimal area and an integral dimension vector. If the rectangles in B can be packed inside some container of area k, there there is a container C^* with dimensions $c_1 \times c_2$, with $c_1, c_2 \in \mathbb{N}$, and $c_1 \cdot c_2 = k$. This choice of dimensions will be generated by the first step of our algorithm. By Lemma 49, for a YES instance, there is an integral packing in a container of minimum area. As we are enumerating all possible ways in which a rectangle edge may pass through a grid point, we effectively enumerate all tilings of a fixed container such that the tiling is an integral packing of the participating tiles. If the given input instance is a YES instance, there is an optimal integral packing P^* can be extended to a tiling T^*, with T as the participating set of tiles and where the corners of each tile has integral co-ordinates: this can be done by adding sufficiently many 1×1 tiles to cover the gaps. The bipartite graph on B and T will have a matching saturating B, since P^* is contained in T^*. On the other hand, if there is a container C^* and a tiling T^* such that there is a matching saturating B in the bipartite graph M_{T^*}, then the matching gives us a packing of B in the container C^*. This shows that the given instance is a YES instance.

Running time: Except for the step of generating configurations, all other steps of the algorithm run in polynomial time (recall that finding a matching in a bipartite graph is polynomial time solvable). The set of configurations that are generated take $(2 \cdot \sqrt{2})^k \cdot k^{O(1)}$ time, for a fixed container. Therefore, the running time of the algorithm is $(2 \cdot \sqrt{2})^k \cdot k^{O(1)}$. □

Theorem 34 can be generalized to higher dimensions. To get dimension vectors for a possible optimal container we try all factorizations of k with d factors in it. To avoid any ambiguity, we redefine the analogues of structures defined in Theorem 34. A cell in

d-dimensions is a minimal d-dimensional hypercube with unit side length and containing 2^d grid points, all of which are corner points. A face of a cell is a $d-1$ dimensional hypersurface bounding it (corresponding to a rectangle edge in 2-dimensions). To generate a configuration in d-dimensions for $d \geq 2$ we assign each face to be either a solid or a dotted face. The number of faces bounding a d-dimensional cell is $2d$. For a tiling to be valid each boundary face of container is assigned a solid face. To bound the number of internal faces, observe that there are k unit cells in a given bounding box and each face is shared by two of the cells. Hence the number of faces is upper bounded by $k \cdot d$, this gives us a bound of at most $2^{k \cdot d}$ configurations.

Analogous to proof of Theorem 34, we can extract a tiling T from a configuration and check if it gives us a packing. We get the following result.

Theorem 35. *d-p-Volume-Packing for a set of rectangles in $\mathbb{N}^d$ with parameters d and k admits a running time of $2^{k \cdot d} \cdot k^{O(d)}$.*

Theorem 34 gives us the following structural property, which will be useful later.

Corollary 11. *Given a d-dimensional container with volume k, there are at most $2^{k \cdot d}$ configurations, and therefore at most $2^{k \cdot d}$ tilings such that for each tile the corner points have integral coordinates. These configurations and tilings can also be found in $2^{k \cdot d} \cdot k^{O(d)}$ time. For $d = 2$, the number of such configurations and tilings is at most $(2\sqrt{2})^k$ and can be found in time $(2\sqrt{2})^k \cdot k^{O(1)}$.*

6.4 Strip Packing

In this section, we will study the parameterized complexity of Strip Packing problem. This problem is already known to be NP-Hard [71]. We first show that Strip Packing in its general form is W[1]-hard, even when the input is represented in unary form.

Lemma 51. *d-p-Strip Packing is W[1]-hard for $d \geq 2$.*

Proof. We prove the result by giving a reduction from Bin Packing problem to the d-p-Strip Packing problem. An instance of the Bin Packing problem takes in as input a set of n objects $a_1, a_2, \ldots, a_n$, with each object a_i weighing w_i, and positive integers W and k. The goal is to decide whether all the input objects can be fit into at most k bins, each of capacity W. It was shown in [60] that Bin Packing is W[1]-hard, even when the the input is represented in unary form.

Let $(\{a_1, a_2, \ldots, a_n\}, \{w_1, w_2, \ldots, w_n\}, W, k)$ be an instance of Bin Packing. We construct an instance of d-p-Strip Packing as follows:

1. First we construct a vector $W' \in \mathbb{N}^{d-1}$. The vector has W in its first index and 1 at all other indices.

2. Next, we construct a set of n boxes. For each object a_i, we create a box b_i whose dimension vector in $\mathbb{N}^d$ has w_i in the first index, and 1 in all other indices.

This completes the construction of our reduced instance. We claim that the Bin Packing is a Yes instance if and only if the reduced instance of d-p-Strip Packing is a Yes instance.

First, suppose the given instance of Bin Packing is a Yes instance. This means that at most k bins of capacity W are sufficient to pack all the input objects. In the d-p-Strip Packing instance, we construct the following arrangement for the input boxes. For $1 \leq j \leq k$, let $\{a_{i_1}, a_{i_2}, \ldots, a_{i_j}\}$ be the set of objects that are placed in the j^{th} bin. Then, for each $1 \leq \ell \leq i_j$, we place the box b_ℓ such that the left-most bottom-most coordinate is at $(\Sigma_{1 \leq \ell' < \ell} w_{\ell'}, 1^{d-2}, j-1)$. By the construction, for a bin j, all the boxes, corresponding to the objects in bin j, can be placed without overlap in such a way that their right-most top-most coordinate is at $(\Sigma_{1 \leq \ell' \leq \ell} w_{\ell'}, 1^{d-2}, j)$. Also, when $j_1 \neq j_2$, then a box corresponding to an object in bin j_1 does not overlap with a box corresponding to an object in bin j_2. This means that all boxes can be packed within a height of k of the given container, whose left-most bottom-most point is placed at the origin.

In the reverse direction, let the constructed d-p-Strip Packing instance be a Yes instance.

By Lemma 49, we know that there is a witnessing packing such that all the input boxes are placed in integral coordinated in an arrangement that requires the height of the box to be at most k. Since the height of each input box is 1, an integral arrangement would mean that the last coordinate for the base and the top of each box is an integer. Thus, the container of dimension $W' \times k$ can be divided into k smaller containers, each of dimension $W' \times 1$. Let $\{b_{i_1}, b_{i_2}, \ldots, b_{i_j}\}$ be the boxes in the j^{th} subcontainer. Then, we place the objects $\{a_{i_1}, a_{i_2}, \ldots, a_{i_j}\}$ in the j^{th} bin for the Bin Packing instance. By construction, in each bin, the sum of weights of objects packed is at most W. Therefore, we require at most k bins to pack all given objects, thereby showing that the given Bin Packing instance is a Yes instance.

This completes the proof of W[1]-hardness of d-p-Strip Packing. □

Now we consider a special case where Strip Packing becomes FPT. This special case is motivated by the variant of the Bin Packing problem, where the number of types of weights is bounded by a constant. Determining the complexity of this problem was a long standing open problem. In [47], the problem was shown to be polynomial time solvable. We study the case where the dimension of each rectangle is bounded by a constant ℓ. This also implies that the types of rectangles are bounded by ℓ^2.

Lemma 52. 2-Strip Packing *is FPT when the dimensions of the input rectangles belong to a set S such that $\ell = \max_{a \in S} a$ is a constant. The running time of the algorithm is* $8^{k\ell} \cdot (n^{O(\ell^2)} W)$

Proof. There are at most ℓ^2 shapes of boxes that have their dimensions in the set S. Assume that among the n input boxes, n_i boxes of shape $i \leq \ell^2$ are there. Let this be denoted by the vector $(n_1, n_2, \ldots, n_{\ell^2})$.

Consider a packing of rectangles in a strip of width W and height k. Assume this is a perfect packing. We will consider vertical layers of fixed width in this packing. The first layer consists of all rectangles whose right edge is at most ℓ distance away from the left

boundary of the strip. The second layer consists of rectangles whose right edge is at a distance of at most ℓ from the rightmost point of the first layer and so on. Since this is a perfect packing, for each layer, there is a bounding box of dimension $2\ell \times k$. Note that a bounding box can intersect with rectangles in the previous and next layers also. The packing contains at most W such layers. Moreover, by the bound on the number of configurations given in Corollary 11, the number of possibilities for the pattern for each layer is a $8^{k\ell}$. Let $\mathcal{P}$ be the set of all $8^{k\ell}$ such patterns. We build a directed graph, G as follows. Each vertex v in $V(G)$ corresponds to a pattern P_v and there exists a directed edge (u, v), from u to v, if P_u can be immediately followed by P_v i.e., for each rectangle $r_1 \in P_v$ whose left edge intersects with the left boundary of P_v there is a rectangle $r_2 \in P_u$ whose the right boundary intersects with the right boundary of P_u and the left boundary of r_1. The weight of edge (u, v) is a vector $((t_1^v, t_2^v, \ldots, t_{\ell^2}^v), \mathsf{dist}_{uv})$, where t_i^v is the number of boxes of type i in the pattern P_v, and dist_{uv} is the distance(l_1 norm in X axis) between the rightmost point on the right boundary of P_u and the rightmost point on the right boundary of P_v. Now, suppose there is a walk in the graph G of total weight $((N_1, N_2, \ldots, N_{\ell^2}), W')$, such that for each i, $N_i \geq n_i$ and $W' \leq W$. Then this corresponds to an arrangement of boxes in a container of dimension $W \times k$. On the other hand, if there is an arrangement of boxes in a container of dimension $W \times k$, we can always construct a walk in G that has weight $((N_1, N_2, \ldots, N_{\ell^2}), W')$, such that for each i, $N_i \geq n_i$ and $W' \leq W$.

What remains is to find such a walk if it exists, or correctly detect that the instance is a No instance. For this, we design a dynamic programming algorithm. For each vector $(M_1, M_2, \ldots M_{\ell^2})$, where each $M_i \leq n_i$, for each $W' \leq W$ and for each pattern P_v, we want to determine whether there is a walk of weight $((M_1, M_2, \ldots M_{\ell^2}), W')$ such that the last vertex in the walk is v. In the base case, it is trivial to update entries of walks with weight 1. As an induction hypothesis, let us suppose that for each vector $(M_1, M_2, \ldots M_{\ell^2})$, where each $M_i \leq n_i$, for each $W'' < W'$ and for each pattern P_v, we have determined whether there is a walk of weight $((M_1, M_2, \ldots M_{\ell^2}), W'')$ such that the last vertex in the walk is v. To determine whether there is a walk of weight $((M_1, M_2, \ldots M_{\ell^2}), W')$ such that the

last vertex in the walk is v, it is enough to determine whether there is a in-neighbor u of v such that there is a walk of weight $((M_1 - t_1^v, M_2 - t_2^v, \ldots M_{\ell^2} - t_{\ell^2}^v), W' - \mathsf{dist}_{uv})$ such that the last vertex in the walk is u. By definition, $W' - \mathsf{dist}_{uv} < W'$. Therefore, by induction hypothesis, if such a walk ending at u exists then we have stored the answer correctly. If such a walk exists for an in-neighbor u of v, this also implies that there is a walk of weight $((M_1, M_2, \ldots M_{\ell^2}), W')$ such that the last vertex in the walk is v. Hence, in polynomial time, we can update each entry of the table. In the end, we search over all vertices $v \in V(G)$, and all weights $W' \leq W$ whether there is a walk of weight $((M_1, M_2, \ldots M_{\ell^2}), W')$ such that the last vertex in the walk is v, and for each i, $M_i \geq n_i$.

The size of the table is $n^{\ell^2} \cdot W \cdot 8^{k\ell}$. Each entry takes polynomial time to be updated. In the end, searching for a required walk also needs polynomial time. Thus, we have given an algorithm with running time $8^{k\ell} \cdot (n^{O(\ell^2)}W)$, which is an FPT algorithm parameterized by k, when ℓ is a constant. □

6.5 Polynomial time packings

In this section, we look at some special cases, where square boxes of specific types are taken. Moreover, the dimension of a smaller square divides that of a larger square. For these cases, we give combinatorial arguments to show that an optimal packing can be found in polynomial time. First, we consider the case when 1×1 squares are given as input.

Lemma 53. *Given a container whose width is a positive integer W, n_1 boxes of width 1 and height 1, and n_2 boxes of width l and height l, we can determine in polynomial time the minimum height required such that all input boxes can be packed into the given container.*

Proof. We use a greedy approach where all the bigger rectangles are packed first and the

remaining space is filled with 1×1 rectangles. n_2 rectangles of dimension $l \times l$ are packed first as tightly as possible with each row (except the top most) containing $\lfloor \frac{W}{l} \rfloor$ rectangles. Next, we pack the 1×1 rectangles starting with any row that is not filled and continuing till we pack all rectangles. We claim that this packing optimizes the height used.

Let k_1 be the height to which $l \times l$ rectangles are packed and k_2 be the height to which 1×1 rectangles are packed. If $k_1 \geq k_2$, then this is an optimum packing since packing only $l \times l$ rectangles needs a height of k_1 in the strip. Otherwise, there is no gap till height $k_2 - 1$. This implies that k_2 is the optimum height since the combined area of the input rectangles, $A = n_1 + n_2 l^2$, is such that $W \cdot (k_2 - 1) < A \leq W \cdot k_2$. □

We can generalize Lemma 53 to the following.

Lemma 54. *Given a container of width W, n_1 boxes of width l and height l, and n_2 boxes of width cl and height cl,where c is an integer, we can determine in polynomial time the minimum height required such that all input boxes can be packed into the given container.*

Proof. We use a greedy algorithm as before and pack all the bigger rectangles first and later fill up the gaps with the smaller rectangles. Let k_1 be the height to which $l \times l$ rectangles are packed and k_2 be the height to which $cl \times cl$ rectangles are packed. If in this packing, the rectangles are packed without leaving any gaps upto a height k', then the packing is optimum by reasons given in the proof of Lemma 53. Otherwise, either all the small rectangles are packed in one layer in which case this packing is optimum since k_2 is a lower bound on the height needed for packing all big rectangles. Or the rectangles are packed to a width $W' > W - l$. Consider an optimal packing of the boxes. We show that for any optimal packing, at each integer row of the container, the total width covered by the rectangles intersecting with the row is at least $W - W'$. This is because $W - W'$ is the smallest remainder when dividing W by l. Therefore, for any packing of height k in a container of width W, there exists a packing of height k in a container of width W'. By the arguments of Lemma 53, the packing is optimum for width W'. This implies that the

packing is optimum for width W. □

Finally, we can give a polynomial packing for the following case.

Lemma 55. *We are given a container of width W, and integers $a_1 < a_2 .. \ldots < a_t$ such that for each $i \leq t$, a_i is a divisor for all a_j, $j > i$. Let there be n_i boxes of width a_i and height a_i. We can determine in polynomial time the minimum height required such that all the input boxes can be packed into the given container.*

Proof. We give a packing for these squares as follows: We start with the largest available square and place it as low and left as possible. We continue this process till all squares have been packed. We claim that this is an optimal packing for the given set of squares.

We prove the optimality of this packing by induction on t. The base case, when $t = 2$, is true due to Lemma 54. Suppose the algorithm finds an optimal packing when the container has width W, the dimension vectors of the input squares are $\{b_1 \times b_1, b_2 \times b_2, \ldots b_{t-1} \times b_{t-1}\}$, and for each $i \leq t-1$, b_i is a divisor for all b_j, $j > i$. Now, we try to give a packing for the input set of squares, having dimensions $a_j \times a_j$, $j \geq 1$. First, by induction hypothesis, we use our algorithm to pack the set of squares with dimension vectors $a_i \times a_i$, $i > 1$. There are $t-1$ distinct dimension vectors. Therefore, our algorithm gives an optimal packing of these squares. Since this is an optimal packing of these squares, the height k_1 required for this packing is a lower bound for any optimal packing of the original set of squares. Now we try to add the squares of dimension $a_1 \times a_1$ according to the algorithm. Suppose the height taken after packing the $a_1 \times a_1$ squares is k_1 then we know that this is an optimal packing. Otherwise, let the height required be $k > k_1$. Except for the top row(which contains only squares of size $a_1 \times a_1$), all other rows incur a gap which is strictly less that a_1. As argued in Lemma 54, for any packing of the input boxes, all rows except the top row incur at least as much gap as our packing. Let the sum of all the gaps in all but the top row be G. By arguments similar to Lemma 53, the total sum of the areas of the squares and G is strictly greater than $W(k-1)$. Therefore, k is the optimum height. □

Chapter 7

Conclusion and Open Problems

In chapters 2 and 3 we studied the vertex deletion and edge deletion problems of reducing the "rank of the graph" for undirected and oriented graphs. We saw that even though the considered vertex and edge modification problems are NP-Complete they still admit FPT algorithms and kernels. We left a few problems open. Firstly, is it possible to obtain improved kernels and algorithms for these problems? In particular, is it possible to improve the exponent of the subexponential algorithm for r-Rank Edge Deletion and r-Rank Editing (in the case of undirected graphs)? It would be interesting to get rid of the $\log k$ factor in the exponent of running time for edge deletion and editing algorithms. Secondly, can the bound for maximum number of vertices in a reduced directed graph of rank r be improved? In the edge editing version of the problem on oriented graphs, a natural direction to pursue from here would be to find if the problem r-Rank Editing is NP-Complete. We note that our hardness reduction does not seem to generalize easily to the editing version. Furthermore, what is complexity of the problem of reducing the number of distinct eigenvalues of a graph by deleting a few vertices or editing a few edges?

In chapter 4 we generalize the results in chapters 2 and 3. In chapter 4 we initiated the study of parameterized graph modification problems in general, and the parameterized Matrix Rigidity problem in particular, in general matrices. First, we used a chain of

reductions: from a special case of Odd Set to a variant of Odd Set that we called $\mathbb{F}$-Odd Matrix, from $\mathbb{F}$-Odd Matrix to a variant of Nearest Codeword that we called $\mathbb{F}$-Nearest Codeword, and finally from $\mathbb{F}$-Nearest Codeword to Matrix Rigidity. Thus, we showed that both Real Matrix Rigidity and FF Matrix Rigidity over any finite field are W[1]-hard with respect to k.

Second, we showed that an input instance of Matrix Rigidity can be pruned to obtain an equivalent instance in which the matrix has $O(r^2 \cdot k^2)$ entries. To this end, we employed an algorithm Column-Reduction which relies on carefully decomposing an input matrix into blocks of linearly independent columns. Then, we turned to show that both Real Matrix Rigidity and FF Matrix Rigidity are FPT with respect to $r + k$. For this purpose, we reduced the problem to checking the feasibility of several sets of polynomials over a field. For both these we obtained algorithms running in polynomial time in the bit-lengths of the entries of the input matrices. Finally, we carefully adapted the technique used to design the algorithm for r-Rank Edge Deletion in chapter 3 to the study of matrices over a finite field, and thus developed an algorithm that runs in time $O^*(2^{O(f(r,p)\sqrt{k}\log k)})$.

We left several problems open in chapter 4. First, we find it interesting to study the parameterized complexity of Matrix Rigidity over rationals. Another intriguing question asks whether Matrix Rigidity admits a parameterized algorithm with respect to $k + r$ that has a subexponential dependency on k, not allowing an exponential dependency on the size of the field. In this context, it also seems non-trivial to significantly improve the dependency on r or to shave the $\log k$ factor that are part of the O^*-running time of the algorithm we gave in Section 4.6. Finally, we note that many natural parameterized matrix modification problems remain to be explored, such as editing to totally unimodular matrices or to matrices representing certain matroids.

In chapter 5 we continued on the theme of using the concept of rank in parameterized complexity and presented a (randomized) polynomial compression of the Vertex Cover Above LP problem into the algebraic Rank Vertex Cover problem. With probability at

least $1 - \varepsilon$, the output instance is equivalent to the original instance and it is of bit length $\tilde{O}(k^7 + k^{4.5} \log(1/\varepsilon))$. Here, the probability ε is part of the input. Recall that having our polynomial compression at hand, one also obtains polynomial compressions of additional well-known problems, such as the Odd Cycle Transversal problem, into the Rank Vertex Cover problem. We note that we do not know how to derandomize our polynomial compression, and it is also not known how to derandomize the polynomial kernelization by Kratsch and Wahlström [66]. The following intriguing open problem still remains unsolved: Does there exist a deterministic polynomial compression of the Vertex Cover Above LP problem?

In chapter 6 we studied Strip Packing and Volume Packing in the parameterized setting, when the inputs are boxes with integral dimension vectors, and the containers are also boxes with integral dimension vectors. We showed that d-p-Volume Packing is FPT, but d-Strip Packing is W[1]-hard. Because of extensive applications of these problems, it is also interesting to study special variants of these problems. We studied 2-Strip Packing with bounded dimensions. This problem is motivated from the study of Bin Packing with bounded types. An open question is regarding the status of 2-Strip Packing with bounded types. In other words, if the number of types of input boxes was a constant, then what is the complexity of the problem? The same can be asked for d-Strip Packing with bounded types. Lastly, we saw some polynomial time optimal packings for some special cases of Strip Packing where the input boxes are squares with dimension vectors satisfying certain constraints. It would be interesting to study these packing problems when the input boxes are arbitrary squares.

Bibliography

[1] Abu-Khzam, F.N.: A kernelization algorithm for d-hitting set. Journal of Computer and System Sciences 76(7), 524–531 (2010)

[2] Akbari, S., Cameron, P.J., Khosrovshahi, G.B.: Ranks and signatures of adjacency matrices (2004)

[3] Alt, H., Berg, M., Knauer, C.: Algorithms - ESA 2015: 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings, chap. Approximating Minimum-Area Rectangular and Convex Containers for Packing Convex Polygons, pp. 25–34. Springer Berlin Heidelberg, Berlin, Heidelberg (2015)

[4] Alt, H., Scharf, N.: Approximating smallest containers for packing three-dimensional convex objects. CoRR abs/1601.04585 (2016), http://arxiv.org/abs/1601.04585

[5] Ashok, P., Kolay, S., Meesum, S., Saurabh, S.: Parameterized complexity of strip packing and minimum volume packing. Theoretical Computer Science pp. – (2016)

[6] Balasubramanian, R., Fellows, M.R., Raman, V.: An improved fixed-parameter algorithm for vertex cover. Information Processing Letters 65(3), 163–168 (1998)

[7] Basu, S., Pollack, R., Roy, M.F.: Algorithms in Real Algebraic Geometry (Algorithms and Computation in Mathematics). Springer-Verlag New York, Inc., Secaucus, NJ, USA (2006)

[8] Beasley, J.E.: An exact two-dimensional non-guillotine cutting tree search procedure. Operations Research 33(1), 49–64 (1985)

[9] Bliznets, I., Cygan, M., Komosa, P., Mach, L., Pilipczuk, M.: Lower bounds for the parameterized complexity of minimum fill-in and other completion problems. In: Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 1132–1151. Society for Industrial and Applied Mathematics (2016)

[10] Bliznets, I., Fomin, F.V., Pilipczuk, M., Pilipczuk, M.: A subexponential parameterized algorithm for proper interval completion. SIAM Journal on Discrete Mathematics 29(4), 1961–1987 (2015)

[11] Bliznets, I., Fomin, F.V., Pilipczuk, M., Pilipczuk, M.: Subexponential parameterized algorithm for interval completion. In: Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 1116–1131. Society for Industrial and Applied Mathematics (2016)

[12] Bonnet, E., Egri, L., Marx, D.: Fixed-parameter approximability of boolean MinCSPs. In: ESA. pp. 18:1–18:18 (2016)

[13] Brown, A.: Optimum Packing and Depletion: The Computer in Space - And Resource-Usage Problems. Computer monographs, 14, MacDonald (1971)

[14] Burzyn, P., Bonomo, F., Durán, G.: Np-completeness results for edge modification problems. Discrete Applied Mathematics 154(13), 1824–1844 (2006)

[15] Buss, J.F., Goldsmith, J.: Nondeterminism within p. SIAM J. Comput. 22(3), 560–572 (Jun 1993)

[16] Cai, L.: Fixed-parameter tractability of graph modification problems for hereditary properties. Information Processing Letters 58(4), 171–176 (1996)

[17] Cao, Y.: Unit interval editing is fixed-parameter tractable. In: ICALP. Lecture Notes in Computer Science, vol. 9134, pp. 306–317. Springer (2015)

[18] Cao, Y.: Linear recognition of almost interval graphs. In: Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 1096–1115. Society for Industrial and Applied Mathematics (2016)

[19] Cao, Y., Marx, D.: Chordal editing is fixed-parameter tractable. In: STACS. pp. 214–225 (2014)

[20] Cao, Y., Marx, D.: Interval deletion is fixed-parameter tractable. ACM Transactions on Algorithms (TALG) 11(3), 21 (2015)

[21] Cavers, M., Cioabă, S., Fallat, S., Gregory, D., Haemers, W., Kirkland, S., McDonald, J., Tsatsomeros, M.: Skew-adjacency matrices of graphs. Linear Algebra and its Applications 436(12), 4512–4529 (2012)

[22] Chandran, L.S., Grandoni, F.: Refined memorization for vertex cover. Information Processing Letters 93(3), 125–131 (2005)

[23] Chen, J., Fernau, H., Kanj, I.A., Xia, G.: Parametric duality and kernelization: Lower bounds and upper bounds on kernel size. SIAM J. Comput. 37(4), 1077–1106 (Nov 2007)

[24] Chen, J., Kanj, I.A., Jia, W.: Vertex cover: Further observations and further improvements. Journal of Algorithms 41(2), 280 – 301 (2001)

[25] Chen, J., Kanj, I.A., Xia, G.: Improved upper bounds for vertex cover. Theoretical Computer Science 411(40-42), 3736–3756 (2010)

[26] Coffman, Jr., E.G., Garey, M.R., Johnson, D.S.: Approximation algorithms for np-hard problems. chap. Approximation Algorithms for Bin Packing: A Survey, pp. 46–93. PWS Publishing Co., Boston, MA, USA (1997)

[27] Cygan, M., Fomin, F.V., Kowalik, L., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., Saurabh, S.: Parameterized algorithms. Springer (2015)

[28] Cygan, M., Pilipczuk, M., Pilipczuk, M., Wojtaszczyk, J.O.: On multiway cut parameterized above lower bounds. ACM Trans. Comput. Theory 5(1), 3:1–3:11 (May 2013)

[29] Damaschke, P., Mogren, O.: Editing the simplest graphs. In: International Workshop on Algorithms and Computation. pp. 249–260. Springer (2014)

[30] Dell, H., Van Melkebeek, D.: Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. Journal of the ACM (JACM) 61(4), 23 (2014)

[31] Deshpande, A.J.: Sampling-based algorithms for dimension reduction. Ph.D. thesis, Massachusetts Institute of Technology (2007)

[32] Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer-Verlag (1999), 530 pp.

[33] Downey, R.G., Fellows, M.R.: Fundamentals of Parameterized Complexity. Texts in Computer Science, Springer (2013)

[34] Downey, R.G., Fellows, M.R., Stege, U.: Parameterized complexity: A framework for systematically confronting computational intractability. In: Contemporary trends in discrete mathematics: From DIMACS and DIMATIA to the future. vol. 49, pp. 49–99 (1999)

[35] Drange, P.G., Dregi, M.S., Lokshtanov, D., Sullivan, B.D.: On the threshold of intractability. In: ESA. Lecture Notes in Computer Science, vol. 9294, pp. 411–423. Springer (2015)

[36] Drange, P.G., Fomin, F.V., Pilipczuk, M., Villanger, Y.: Exploring the subexponential complexity of completion problems. ACM Transactions on Computation Theory (TOCT) 7(4), 14 (2015)

[37] Drange, P.G., Pilipczuk, M.: A polynomial kernel for trivially perfect editing. In: ESA. Lecture Notes in Computer Science, vol. 9294, pp. 424–436. Springer (2015)

[38] Eves, H.W.: Elementary matrix theory. Courier Corporation (1966)

[39] Fomin, F.V., Kratsch, S., Pilipczuk, M., Pilipczuk, M., Villanger, Y.: Tight bounds for parameterized complexity of cluster editing with a small number of clusters. Journal of Computer and System Sciences 80(7), 1430–1447 (2014)

[40] Fomin, F.V., Lokshtanov, D., Meesum, S.M., Saurabh, S., Zehavi, M.: Matrix rigidity from the viewpoint of parameterized complexity. In: 34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany. pp. 32:1–32:14 (2017)

[41] Fomin, F.V., Lokshtanov, D., Misra, N., Saurabh, S.: Planar F-deletion: Approximation, kernelization and optimal FPT algorithms. In: FOCS (2012)

[42] Fomin, F.V., Villanger, Y.: Subexponential parameterized algorithm for minimum fill-in. SIAM Journal on Computing 42(6), 2197–2216 (2013)

[43] Friedman, J.: A note on matrix rigidity. Combinatorica 13(2), 235–239 (1993)

[44] Fujito, T.: A unified approximation algorithm for node-deletion problems. Discrete Appl. Math. pp. 213–231 (September)

[45] Garg, S., Philip, G.: Raising the bar for vertex cover: Fixed-parameter tractability above A higher guarantee. In: Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016,. pp. 1152–1166 (2016)

[46] Ghosh, E., Kolay, S., Kumar, M., Misra, P., Panolan, F., Rai, A., Ramanujan, M.S.: Faster parameterized algorithms for deletion to split graphs. Algorithmica 71(4), 989–1006 (2015)

[47] Goemans, M.X., Rothvoß, T.: Polynomiality for bin packing with a constant number of item types. In: Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 830–839. SODA '14, SIAM (2014)

[48] Gordon, G., McNulty, J.: Matroids: A Geometric Introduction. Cambridge University Press (2012), cambridge Books Online

[49] Grigoriev, D.: Using the notions of separability and independence for proving the lower bounds on the circuit complexity (in russian). Notes of the Leningrad branch of the Steklov Mathematical Institute, Nauka (1976)

[50] Grigoriev, D.: Using the notions of separability and independence for proving the lower bounds on the circuit complexity. Journal of Soviet Math. 14(5), 1450–1456 (1980)

[51] Guo, J.: A more effective linear kernelization for cluster editing. In: Combinatorics, Algorithms, Probabilistic and Experimental Methodologies. Springer (2007)

[52] Gutin, G., Kim, E.J., Lampis, M., Mitsou, V.: Vertex cover problem parameterized above and below tight bounds. Theory of Computing Systems 48(2), 402–410 (2011)

[53] Harren, R., Kern, W.: Improved lower bound for online strip packing. Theory Comput. Syst. 56(1), 41–72 (2015)

[54] Hill, R.: A First Course in Coding Theory. Oxford Applied Linguistics

[55] Hoffman, K., Kunze, R.: Linear algebra. 1971. Englewood Cliffs, New Jersey

[56] Huang, E., Korf, R.E.: Optimal rectangle packing: An absolute placement approach. J. Artif. Int. Res. 46(1), 47–87 (Jan 2013)

[57] Hüffner, F., Komusiewicz, C., Moser, H., Niedermeier, R.: Fixed-parameter algorithms for cluster vertex deletion. Theory of Computing Systems 47(1) (2010)

[58] Impagliazzo, R., Paturi, R., Zane, F.: Which problems have strongly exponential complexity? Journal of Computer and System Sciences 63, 512–530 (2001)

[59] Imreh, C.: Online strip packing with modifiable boxes. Operations Research Letters 29(2), 79 – 85 (2001)

[60] Jansen, K., Kratsch, S., Marx, D., Schlotter, I.: Bin packing with fixed number of bins revisited. Journal of Computer and System Sciences 79(1), 39 – 49 (2013)

[61] Jukna, S.: Extremal combinatorics: with applications in computer science. Springer Science & Business Media (2011)

[62] Kayal, N.: Solvability of a system of bivariate polynomial equations over a finite field. In: ICALP. pp. 551–562 (2005)

[63] Kenyon, C., Rémila, E.: A near-optimal solution to a two-dimensional cutting stock problem. Math. Oper. Res. 25(4), 645–656 (Nov 2000)

[64] Kotlov, A., Lovász, L.: The rank and size of graphs. Journal of Graph Theory 23(2), 185–189 (1996)

[65] Kratsch, S.: A randomized polynomial kernelization for vertex cover with a smaller parameter. In: 24th Annual European Symposium on Algorithms, ESA 2016. pp. 59:1–59:17

[66] Kratsch, S., Wahlström, M.: Representative sets and irrelevant vertices: New tools for kernelization. In: Foundations of Computer Science (FOCS), 2012 IEEE 53rd Annual Symposium on. pp. 450–459. IEEE (2012)

[67] Kumar, A., Lokam, S.V., Patankar, V.M., Sarma, M.N.J.: Using elimination theory to construct rigid matrices. Computational Complexity 23(4), 531–563 (2013)

[68] Lampis, M.: A kernel of order $2k - c \log k$ for vertex cover. Information Processing Letters 111(23), 1089–1091 (2011)

[69] Lange, K.: Hadamard's determinant inequality. The American Mathematical Monthly 121(3), 258–259 (2014)

[70] Lewis, J.M., Yannakakis, M.: The node-deletion problem for hereditary properties is np-complete. Journal of Computer and System Sciences 20(2), 219–230 (1980)

[71] Lodi, A., Martello, S., Monaci, M.: Two-dimensional packing problems: A survey. European Journal of Operational Research 141(2), 241 – 252 (2002)

[72] Lokam, S.V.: Spectral methods for matrix rigidity with applications to size-depth tradeoffs and communication complexity. In: FOCS. pp. 6–15 (1995)

[73] Lokam, S.V.: On the rigidity of Vandermonde matrices. Theoretical Computer Science 237(1–2), 477 – 483 (2000)

[74] Lokam, S.V.: Complexity lower bounds using linear algebra. Found. Trends Theor. Comput. Sci. 4, 1–155 (Jan 2009)

[75] Lokshtanov, D., Narayanaswamy, N., Raman, V., Ramanujan, M., Saurabh, S.: Faster parameterized algorithms using linear programming. ACM Transactions on Algorithms (TALG) 11(2), 15 (2014)

[76] Lovász, L.: Flats in matroids and geometric graphs. Combinatorial Surveys pp. 45–86 (1977)

[77] Lund, C., Yannakakis, M.: On the hardness of approximating minimization problems. J. ACM 41, 960–981 (September 1994)

[78] Mahajan, M., Sarma M.N., J.: On the complexity of matrix rank and rigidity. In: CSR. pp. 269–280 (2007)

[79] Marx, D., O'Sullivan, B., Razgon, I.: Finding small separators in linear time via treewidth reduction. ACM Transactions on Algorithms 9(4), 30 (2013)

[80] Meesum, S., Misra, P., Saurabh, S.: Reducing rank of the adjacency matrix by graph modification. Theoretical Computer Science 654, 70 – 79 (2016), computing and Combinatorics

[81] Meesum, S.M., Saurabh, S.: Rank reduction of oriented graphs by vertex and edge deletions. Algorithmica pp. 1–20 (2017)

[82] Mishra, S., Raman, V., Saurabh, S., Sikdar, S., Subramanian, C.R.: The complexity of könig subgraph problems and above-guarantee vertex cover. Algorithmica 61(4), 857–881 (2011)

[83] Murata, H., Fujiyoshi, K., Nakatake, S., Kajitani, Y.: Vlsi module placement based on rectangle-packing by the sequence-pair. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 15(12), 1518–1524 (Dec 1996)

[84] Narayanaswamy, N., Raman, V., Ramanujan, M., Saurabh, S.: Lp can be a cure for parameterized problems. In: STACS'12 (29th Symposium on Theoretical Aspects of Computer Science). vol. 14, pp. 338–349. LIPIcs (2012)

[85] Niedermeier, R., Rossmanith, P.: Upper bounds for vertex cover further improved. In: Annual Symposium on Theoretical Aspects of Computer Science. pp. 561–570. Springer Berlin Heidelberg (1999)

[86] Oxley, J.G.: Matroid Theory (Oxford Graduate Texts in Mathematics). Oxford University Press, Inc., New York, NY, USA (2006)

[87] Peeters, R.: The maximum edge biclique problem is np-complete. Discrete Applied Mathematics 131(3), 651–654 (2003)

[88] Qu, H., Yu, G.: Bicyclic oriented graphs with skew-rank 2 or 4. Applied Mathematics and Computation 258, 182–191 (2015)

[89] Raman, V., Ramanujan, M., Saurabh, S.: Paths, flowers and vertex cover. In: European Symposium on Algorithms. pp. 382–393. Springer Berlin Heidelberg (2011)

[90] Razborov, A.A.: On rigid matrices. Manuscript in russian (1989)

[91] Razgon, I., O'Sullivan, B.: Almost 2-sat is fixed-parameter tractable. Journal of Computer and System Sciences 75(8), 435–450 (2009)

[92] Reed, B., Smith, K., Vetta, A.: Finding odd cycle transversals. Operations Research Letters 32(4), 299–301 (2004)

[93] Renegar, J.: On the computational complexity and geometry of the first-order theory of the reals. part i: Introduction. preliminaries. the geometry of semi-algebraic sets. the decision problem for the existential theory of the reals. Journal of symbolic computation 13(3), 255–299 (1992)

[94] Robertson, N., Seymour, P.D.: Graph minors. xiii. the disjoint paths problem. Journal of combinatorial theory, Series B 63(1), 65–110 (1995)

[95] Sarma M.N., J.: Complexity Theoretic Aspects of Rank, Rigidity and Circuit Evaluation. Ph.D. thesis, The Institute of Mathematical Sciences, CIT Campus, Taramani, Chennai (2009)

[96] Shamir, R., Sharan, R., Tsur, D.: Cluster graph modification problems. Discrete Appl. Math. 144(1-2) (2004)

[97] Shokrollahi, M.A., Spielman, D., Stemann, V.: A remark on matrix rigidity. Information Processing Letters 64(6), 283 – 285 (1997)

[98] Sigler, L.: Algebra. Undergraduate Texts in Mathematics, Springer-Verlag (1976)

[99] Valiant, L.G.: Graph-theoretic arguments in low-level complexity. In: MFCS. pp. 162–176 (1977)

[100] Xia, Y., Chrzanowska-Jeske, M., Wang, B., Jeske, M.: Using a distributed rectangle bin-packing approach for core-based soc test scheduling with power constraints. In: Computer Aided Design, 2003. ICCAD-2003. International Conference on. pp. 100–105 (Nov 2003)

[101] Yannakakis, M.: Node-and edge-deletion np-complete problems. In: STOC. pp. 253–264. ACM (1978)

[102] Ye, D., Han, X., Zhang, G.: A note on online strip packing. Journal of Combinatorial Optimization 17(4), 417–423 (2008)