# Parameterized Complexity of Fair Feedback Vertex Set Problem

**A Thesis**

submitted to
Indian Institute of Science Education and Research Pune
in partial fulfillment of the requirements for the
BS-MS Dual Degree Programme

by

Komal Muluk


Indian Institute of Science Education and Research Pune
Dr. Homi Bhabha Road,
Pashan, Pune 411008, INDIA.

April, 2019

Supervisors: Dr. Soumen Maity and Prof. Saket Saurabh

# Certificate

This is to certify that this dissertation entitled Parameterized Complexity of Fair Feedback Vertex Set Problem towards the partial fulfilment of the BS-MS dual degree programme at the Indian Institute of Science Education and Research, Pune represents study/work carried out by Komal Muluk at Indian Institute of Science Education and Research under the supervision of Dr. Soumen Maity, Associate Professor, Indian Institute of Science Education and Research, Pune, Department of Mathematics and at The Institute of Mathematical Sciences under the supervision of Prof. Saket Saurabh, Professor, Institute of Mathematical Sciences, Chennai, during the academic year 2018-2019.

Dr. Soumen Maity

Prof. Saket Saurabh

Committee:

Dr. Soumen Maity

Prof. Saket Saurabh

Dr. Philip Geevarghese

To Bunty,

Because he always loved, cared and understood.

# Declaration

I hereby declare that the matter embodied in the report entitled Parameterized Complexity of Fair Feedback Vertex Set Problem are the results of the work carried out by me at the Department of Mathematics, Indian Institute of Science Education and Research, Pune, under the supervision of Dr. Soumen Maity and at The Institute of Mathematical Sciences, Chennai, under the supervision of Prof. Saket Saurabh and the same has not been submitted elsewhere for any other degree.

Komal Muluk

# Acknowledgements

I would like to start by expressing my sincerest gratitude towards Prof. Soumen Maity for his constant support and guidance. I would like to thank him for encouraging and helping me to shape my interest and ideas in the field. I am very grateful for being a part of the group discussions we had. His constant energy and enthusiasm inspired me to work hard.

I would like to thank Prof. Saket Saurabh for providing me with an opportunity to visit The Institute of Mathematical Sciences (IMSc) Chennai. It has been a really great experience working under his supervision. His thorough understanding of the subject together with the crucial discussions helped me gain an in-depth insight into the problem.

I also thank Dr. Philip Geevarghese (CMI) for his valuable time and suggestions related to various strategies for tackling the problem.

I take this opportunity to thank Prof. Venkatesh Raman, for allowing me to attend his course, Parameterized Complexity, at IMSc which enhanced my conceptual understanding of the subject matter.

I thank the Department of Mathematics, IISER Pune for providing a platform for doing such a project and necessary resources required for the same.

I really thank Ashwin and Vibha for their efficient discussions as they were beneficial in achieving a clear approach towards the problem.

Next, I would like to thank Vishnu, Spoorthy and Chinmay, for being inquisitive during my presentations and providing their useful comments. I thank Vaishali for the curious discussions on the problem. I thank Aniruddha and Swati for their valuable suggestions on my first draft of the thesis.

Finally, I would like to thank my parents and my brother for their constant support and motivation. I would like to thank my friends from IISER Pune, Rasika and Vipul for always being there and helping me go through emotional ups and downs throughout the tenure.

# Abstract

Feedback Vertex Set (FVS) problem is a well known NP-complete problem in computational complexity theory. This is a vertex deletion problem in which given a graph $G = (V, E)$ and an integer $k$, we are asked to find a subset $S \subseteq V(G)$ with $|S| \leq k$ for which the remaining graph $G \setminus S$ is a forest (acyclic). In this project, we study the fairness property of Feedback Vertex Set problem. Fair Feedback Vertex Set problem (FFVS) is another vertex deletion problem which also demands to find a set $S \subseteq V(G)$ such that $G \setminus S$ is acyclic, along with an extra condition, that no vertex in the graph should contribute too much to the set $S$. In order to measure the bound on this condition, we give an integer $\ell$, along with the graph $G = (V, E)$ as an input of FFVS problem and we formulate the condition as $|N(v) \cap S| \leq \ell$ for all $v \in V(G)$. Here $N(v) = \{u|(u, v) \in E(G)\}$. The objective here is to equidistribute the solution. This project includes the study of this problem, along with some other variants of this problem, on the realms of parameterized complexity theory.

Parameters which we use throughout this project are the structural graph parameters, they depend on the structure of the input graph provided. We show that FFVS is $W[1]$-hard when parameterized by treewidth and treedepth of the graph. We also prove that it admits an FPT algorithm running in time $\mathcal{O}^*(3^k k^2)$ if parameterized by neighbourhood diversity.

Another problem which we study here is Minimum Fair Feedback Vertex Set (Min-FFVS). This is a restriction of FFVS problem with one more condition applied. In Min-FFVS problem, given a graph $G = (V, E)$ and two positive integers $k$ and $\ell$, the task is to find $S \subseteq V(G)$, $|S| \leq k$ such that $G \setminus S$ is acyclic and all vertices in the graph fulfill the condition that $|N(v) \cap S| \leq \ell$. For Min-FFVS problem, we first show that it is NP-complete. Then we find a kernel of size $\mathcal{O}((\Delta + k)^2)$ using parameter $(\Delta + k)$, where $\Delta$ is the maximum degree of $G$.

Later, we relax one condition and try to find a FFVS which is fair on the remaining graph, that is, $|N(v) \cap S| \leq \ell$ for all $v \in V(G) \setminus S$, if $S$ is the FFVS. This problem, Relaxed Fair Feedback Vertex Set (Relax-FFVS), is tractable if we consider the solution size as a parameter. We get an $\mathcal{O}^*(17^k)$ algorithm for this problem, where $k$ is the size of $S$.

# Contents

**Abstract** **xi**

**1 Introduction** **1**

1.1 Fair Feedback Vertex Set . . . . . 2

1.2 Minimum Fair Feedback Vertex Set . . . . . 4

1.3 Relaxed Fair Feedback Vertex Set . . . . . 4

**2 Preliminaries** **7**

2.1 Graph Theory . . . . . 7

2.2 Definitions of structural graph parameters . . . . . 8

2.3 Introduction to complexity classes P and NP . . . . . 11

**3 Parametrized Complexity** **15**

3.1 Parameterized Problems and Parameterized Algorithms . . . . . 15

3.2 Kernelization . . . . . 16

3.3 W-Hierarchy . . . . . 17

**4 Parameterized Complexity of Fair Feedback Vertex Set Problem** **19**

4.1 On multigraphs, W[1]-hard with respect to treewidth . . . . . 19

4.2 FFVS problem on simple graphs is W[1]-hard with respect to treedepth . . . 23

4.3 FFVS on simple graphs is W[1]-hard with respect to treewidth . . . . . 25

**5 An FPT algorithm for FFVS Problem** **29**

5.1 FPT with respect to the neighbourhood diversity . . . . . 29

**6 Min-FFVS Problem** **33**

6.1 Min-FFVS problem is NP-complete . . . . . 33

6.2 Kernel for Min-FFVS using parameter $\Delta + k$ . . . . . 34

**7 Relax-FFVS Problem** **37**

7.1 FPT when parameterized by solution size . . . . . 37

**8 Conclusion** **43**

# Chapter 1

# Introduction

Feedback Vertex Set problem (FVS) is one of the Karp's 21 NP-complete problems [4]. This problem has been studied extensively in the computational complexity theory as it is one of the fundamental problems in the theory. It is a vertex deletion problem, where we aim to find a set $S \subseteq V$ such that remaining graph $G \setminus S$ is a forest (a graph which does not contain any cycle). Vertex deletion problems are those in which, given a graph $G = (V, E)$ and a property $\pi$; we are asked to find a set $S \subseteq V(G)$ such that the subgraph obtained after the deletion of vertices in $S$, that is, graph $G \setminus S$ satisfies the desired property $\pi$. In FVS problem the desired property is to get an acyclic graph.

In the optimization version of deletion problems (vertex deletion / edge deletion), we focus on minimizing the number of elements require to remove to achieve the desired property on the remaining graph. Many combinatorial optimization problems can be formulated in the form of some deletion problems. For example, Vertex Cover, Feedback Vertex Set, d-Bounded Degree Deletion, Odd Cycle Transversal can be formulated as vertex deletion problem, whereas problems like Perfect Matching, Edge Bipartization can be formulated as edge deletion problem.

Modified versions of deletion problems, called fair deletion problems were introduced by Lin and Sahi in 1989 [5]. Unlike usual deletion problems, fair deletion problems aim to minimize the maximum number of neighbours contributed to the solution set by a single vertex in the graph. Fair vertex deletion problems and fair edge deletion problems are formulated as follows:

Fair Vertex Deletion Problem [2]
**Input:** An undirected graph $G = (V, E)$, a property $\pi$ and a positive integer $\ell$.
**Task:** Is there a set $S \subseteq V(G)$ such that $G \setminus S$ satisfies the property $\pi$ and

$$\max_{v \in V(G)} \{|N(v) \cap S|\} \leq \ell.$$

Fair Edge Deletion Problem [2]
**Input:** An undirected graph $G = (V, E)$, a property $\pi$ and a positive integer $\ell$.
**Task:** Is there a set $F \subseteq E(G)$ such that $G \setminus F$ satisfies the property $\pi$ and for every vertex $v$ in $G$, the number of edges in $F$ incident on $v$ is at most $\ell$.

Now, we define some terminologies which are useful while discussing fair deletion problems. Here, we present them for vertex deletion problems.

**Definition 1.** ***(Fair Cost)*** *Fair cost of a set $S \subseteq V$ is $\max_{v \in V}\{|N(v) \cap S|\}$.*

**Definition 2.** ***(Fair Objective Function)*** *A function that assigns each set $S \subseteq V$ to its fair cost is called a Fair Objective Function.*

## 1.1 Fair Feedback Vertex Set

Dusan Knop, Tomas Masarik and Tomas Toufar defined the Fair Vertex Cover problem in 2018 [3]. They also studied the parameterized complexity of this problem concerning some structural graph parameters. On a similar notion, in this project, we define Fair Feedback Vertex Set (FFVS) problem and study it in the field of parameterized complexity. In Fair Feedback Vertex Set, our objective is to find a feedback vertex set (FVS) such that its cost is not too high for any vertex in the graph.

Fair Feedback Vertex Set (FFVS)
**Input:** An undirected graph $G$ and a positive integer $\ell$.
**Task:** Decide whether there exists a set $S \subseteq V(G)$ such that $G \setminus S$ is a forest and

$$\max_{v \in V(G)} \{|N(v) \cap S|\} \leq \ell.$$

Here, a point to observe is that the size of the desired solution set is not bounded. In fact, the solution size could be arbitrary. The goal here is to find an FVS of any size which satisfies the conditions that the remaining graph is a forest and fair cost of the solution set is bounded by $\ell$. In this problem, we try to equidistribute the solution over all the vertices in the instance.

In the optimization version of FVS problem, we insist on getting the minimum size subset $S \subseteq V$ that leaves the graph $G \setminus S$ acyclic. On the other hand, the optimization version of FFVS problem is to find a feedback vertex set $S \subseteq V$ with the least possible fair cost over all feedback vertex sets of the graph. Here, the main aim is to minimize the fair cost of the solution set.

Now, we illustrate the difference between FVS problem and FFVS problem with the help of the following example:

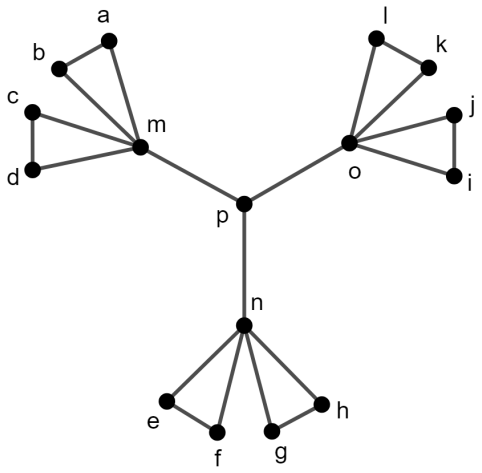



Figure 1.1: Graph $G$

In the graph $G$ shown in Figure 1.1, a minimum FVS of $G$ is the set $S = \{m, n, o\}$, but the fair cost of this set S is 3, as $|N(p) \cap S| = 3$. Set $S$ can't be a solution to FFVS problem when $\ell = 2$, though it is a minimum FVS of $G$. Instead, if we consider another FVS of $G$, say set $S' = \{m, n, k, i\}$, the fair cost of $S'$ is 2. Hence, $S'$ is a solution of FFVS problem on $G$ for $\ell = 2$.

## 1.2 Minimum Fair Feedback Vertex Set

Considering the importance of both the factors, viz., fair cost and solution size, we introduce another version concerning the fairness of FVS problem by adding one more restriction to the FFVS problem. We defined the new problem as follows:

| Minimum Fair Feedback Vertex Set (Min-FFVS) |
|---|
| **Input:** A simple undirected graph $G$ and positive integers $\ell$ and $k$. |
| **Task:** To check whether there exists a set $S \subseteq V(G)$, $\lvert S\rvert \leq k$ such that $G \setminus S$ is an acyclic graph and $\max_{v \in V(G)} \{\lvert N(v) \cap S\rvert\} \leq \ell$. |

It is clear that if there exists a solution for the Min-FFVS problem on graph $G$, then there exists a solution for the classical FVS problem on $G$. This relation does not hold for FFVS problem defined in Section 1.1.

In Min-FFVS problem, we put more restrictions than FVS as well as FFVS problem. Graph $G$ in Figure 1.1 contains an FVS of size-3 which is set $\{m, n, o\}$. It also has an FVS of fair cost-2 which is set $\{m, n, k, i\}$. But $G$ does not contain an FVS of size-3 and fair cost-2 simultaneously. Instead, it contains an FVS of size-3 and fair cost-3, which is again the set $\{m, n, o\}$. Graph $G$ also contains an FVS of size-4 and fair cost-2, which is set $\{m, n, k, i\}$.

## 1.3 Relaxed Fair Feedback Vertex Set

Observing that we put too many restrictions on FFVS and Min-FFVS problem, here we define a relaxed version of the problem.

| Relaxed Fair Feedback Vertex Set (Relax-FFVS) |
|---|
| **Input:** A simple undirected graph $G$ and positive integers $\ell$ and $k$. |
| **Task:** Decide whether there exists a set $S \subseteq V(G)$, $\lvert S\rvert \leq k$ such that $G \setminus S$ is a forest and $\lvert N(v) \cap S\rvert \leq \ell \quad \forall v \in V(G) \setminus S$ |

This way, we don't need to worry about the budget of the vertices which go into the

solution set. Budget is referred to the upper bound on the total number of neighbours a vertex can have in the solution set, that is, integer $\ell$.

In the upcoming chapters of this thesis, we mention some results on the parameterized complexity of FFVS, Min-FFVS and Relax-FFVS problems. Chapter 2 contains some preliminaries in the field of graph theory and introduction to computational complexity theory. It also defines some structural graph parameters which we will use in Chapters 4 and 5. In Chapter 3, we define important notions in parameterized complexity which we further use to prove results in the following chapters after that. In Chapter 4, we prove $W[1]$-hardness of FFVS concerning some parameters. Later, in Chapter 5 we give a parameterized algorithm, precisely, an FPT algorithm to solve the FFVS problem with respect to neighbourhood diversity of the graph. Then we move to Min-FFVS problem in Chapter 6, where we prove that the problem is NP-complete. Next, we obtain a kernel of Min-FFVS problem with respect to parameter $\Delta + k$, where $k$ is the solution size and $\Delta$ is the maximum degree in the graph. At the end we shift towards the study of Relax-FFVS problem. We show that this problem admits an FPT algorithm running in time $\mathcal{O}^*(17^k)$, for $k$ to be the size of the solution set.

# Chapter 2

# Preliminaries

## 2.1 Graph Theory

A graph is a structure which consists of a vertex set $V$ and an edge set $E$ and denoted by $G = (V, E)$. The vertex set $V(G)$ contains the vertices/nodes in the graph and the edge set $E(G)$ is a set of pairs of vertices in $V(G)$. For an edge $e = uv$, also denoted by $\{u, v\}$ or $(u, v)$, where $u, v \in V(G)$, we say $u$ and $v$ are the endpoints of $e$ and $e$ is incident on vertices $u$ and $v$ of graph $G$. We will explain terminologies with the help of the graph $G$ shown in the Figure 2.1. If both the endpoints of an edge are the same vertex, then we call such an edge a loop, e.g., vertex $a$ in $G$. There could also be multiple edges between two vertices of the graph, e.g., two edges between vertices $a$ and $c$ in $G$. Based on the types of edges, there are different types of graphs. The degree of a vertex $v \in V(G)$ is the total number of edges incident on that vertex in $G$. It is denoted by $d(v)$. An isolated vertex is a vertex with degree-0, e.g., vertex $h$ in $G$. $\Delta$ is the maximum degree of a vertex in the graph. Vertices $v$ and $w$ are said to be neighbours in $G$ if $vw \in E(G)$. Neighbourhood of a vertex $v$ in a graph $G$ is $N(v) = \{u \in V(G) | uv \in E(G)\}$.

1. Multigraphs: If a graph contains loops or multiple edges then it is called a multigraph.

2. Simple graphs: It is a loopless graph with no multiple edges.

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

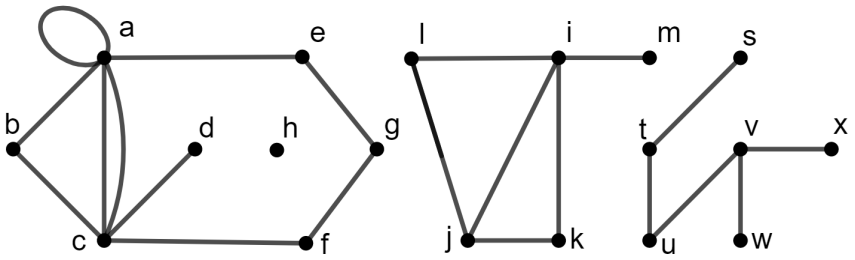



Figure 2.1: Graph $G$

$E'(H) \subseteq E(G)$. $H \subseteq G$ denotes that $H$ is a subgraph of $G$. For a subset $S$ of vertex set of $G$, a subgraph of $G$ induced by $S$, denoted by $G[S]$, is a graph which contains vertex set as $S$ and edge set as a collection of edges whose both endpoints lie in set $S$. For a set $S \subseteq V(G)$, the collection of neighbors of a vertex $v$ in $S$ is denoted by $N_S(v) = \{w \in S | vw \in E(G)\}$. $d_H(v)$ is the total number of neighbours of $v$ in subgraph $H$.

A clique is a set of vertices $W$ such that there are edges between every pair of vertices in $W$. On the other side an independent set is a set of vertices $V$ with no incident edges in $G[V]$. A walk $v_1, e_1, v_2, e_2, \ldots, e_{t-1}, v_t$ is an alternate sequence of vertices and edges, where edge $e_i$ is incident on $v_i$ and $v_{i+1}$. A walk is said to be a closed walk if vertices $v_1$ and $v_t$ are the same. A circuit (cycle) in a graph is a closed walk in which no vertex appears more than once. A path from a vertex $v$ to $u$ is a walk with no repeated vertices or edges in it. A graph $G$ is said to be connected if there exists a path between every pair of vertices in $G$. There are three connected components of graph $G$ shown in Figure 2.1.

A tree is a particular type of graph which is connected and does not contain any cycles. A degree-1 vertex in a tree is called a leaf or a pivot. A collection of trees is called a forest.

## 2.2 Definitions of structural graph parameters

We can measure some graph properties which are purely based on the structure of a graph. Sometimes graph problems can also be studied on the basis of these measures of structural graph properties. Also, these measures can be used as parameters while studying the parameterized complexity of a graph problem ('parameter' and 'parameterized complexity' terms are to be introduced in the next chapter). In this section, we recall definitions of some

structural graph parameters which would be needed for the further discussion throughout this project.

**Definition 3.** ***(Vertex cover)*** *Given a graph $G$, Vertex Cover is the size of the smallest set of vertices $S \subseteq V(G)$ such that after deletion of vertices in $S$, the graph becomes edgeless. It is denoted by $vc(G)$.*

**Definition 4.** ***(Feedback Vertex Set)*** *For a graph $G = (V, E)$, Feedback Vertex Set is the cardinality of the smallest set $S \subseteq V(G)$, such that the graph $G[V \setminus S]$ is a forest. And it is denoted by $fvs(G)$.*

**Definition 5.** ***(Treewidth)*** *Given a graph $G = (V, E)$, treewidth of $G$ is the smallest width of a tree decomposition of $G$ among its all possible tree decompositions. It is denoted by $tw(G)$.*

The width of a tree decomposition is $\max_i(|X_i| - 1)$, where $X_i$'s are bags in a given tree decomposition.

For a graph $G$, a tree decomposition of $G$ is $\tau = (T, \{X_t\}_{t \in V(T)})$, where $T$ is a tree whose vertices are bags. Each bag contains a subset of the vertex set of the original graph $G$ and it satisfies the following properties:

1. Every vertex of the graph should belong to at least one bag of $\tau$.

$$\bigcup_{i \in V(T)} X_i = V(G)$$

2. $\forall\ uv \in E(G), \exists$ at least one bag $X_i$ such that $u, v \in X_i$.

3. For each vertex $u \in V(G)$, the set $\{t \in V(T) | u \in X_t\}$ induces a connected subtree of $T$.

**Definition 6.** ***(Treedepth)*** *Treedepth of a graph $G$ is the minimum height of a rooted forest $F$ whose transitive closure contains the graph $G$. It is denoted by $td(G)$.*

Given a rooted forest $F$, its transitive closure is a graph, say $H$, where $V(H)$ contains all the nodes of the rooted forest and $E(H)$ contain an edge between two vertices only if those two vertices form an ancestor-descendant pair in the forest $F$. We illustrate the idea of a rooted forest, its closure and treedepth using some examples shown in Figure 2.2.

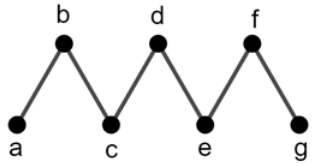
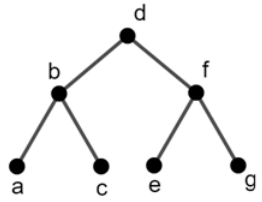
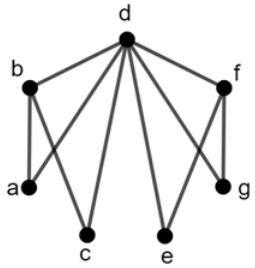
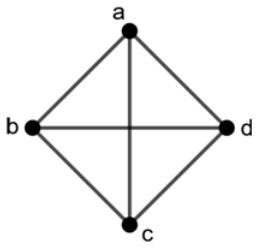
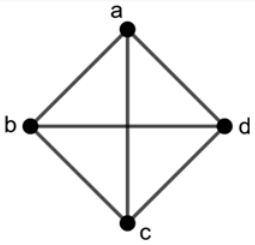
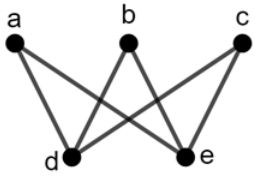
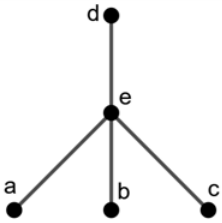
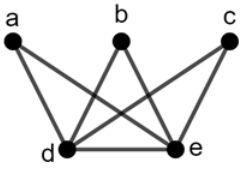
| Ex. No. | Graphs | A Rooted Forest of Minimum Depth | Transitive Closure of the Rooted Forest | td |
|---|---|---|---|---|
| 1. |   |   |   | 3 |
| 2. |   |   |   | 4 |
| 3. |   |   |   | 3 |

Figure 2.2: Examples explaining treedepth of the graph

**Definition 7.** ***(Neighbourhood Diversity)*** *Neighborhood diversity of a graph, denoted by $nd(G)$, is the least integer $k$ for which we can partition the set of vertices of the graph into $k$ classes, which also satisfies the following property:*

*Vertices $u$ and $v$ both belong to the same class if and only if $N(u) \setminus \{v\} = N(v) \setminus \{u\}$.*

Any two vertices which satisfy the property mentioned above are called as twin vertices. Hence, the vertices which belong to the same class in the partition are all pairwise twin vertices. Observe that, each class in this partition either forms a clique or an independent set.

We illustrate the neighbourhood diversity of a graph with the help of an example shown in Figure 2.3. A partition $P = \{\{a, b, c\}, \{e, h, g\}, \{d\}, \{f\}\}$ of $V(G)$ satisfies the condition

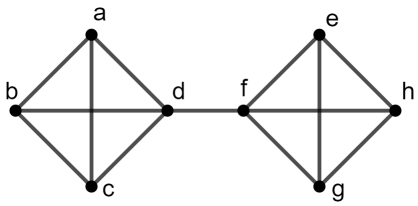



Figure 2.3: Graph $G$

given in the definition. Also, there does not exist a partition of $V(G)$ into less than 4 classes satisfying the property given in the definition. Hence, $nd(G) = 4$.

## 2.3 Introduction to complexity classes P and NP

Complexity theory (lately known as computational complexity theory) concerns with the amount of time and space resources required to solve a computational problem. Time needed for computation is measured in terms of the total number of steps in the performance of the computation, whereas space is the work space (memory storage space) required for the execution of a computation.

Let $\Sigma$ be a set of alphabets and $\Sigma^*$ be the set of strings over alphabets of $\Sigma$. A language is a set of finite strings over some alphabet $\Sigma$. $L$ is a language if $L$ is finite and $L \subseteq \Sigma^*$. A computational problem $P$ is a language $L \subseteq \Sigma^*$. $x$ is said to be a yes-instance of $P$ if and only if $x \in L$. There are various types of computational problems, e.g., decision problems, optimization problems, search problems, etc. For decision problems, the task is only to verify if an input instance satisfies the given property or not. It is a yes/no question.

In theoretical computer science, we are mostly interested in the lower bounds of problems, that is, to show that a certain computational problems cannot be solved in a specific time and space resources. They are too difficult for us to find an algorithm which can be implemented in given conditioned resources. Computational problems are further classified into different complexity classes, according to the difficulty level of the problems.

The main objective of computational complexity theory is to find feasible algorithms to solve a problem. An algorithm is said to be 'feasible' if it runs in polynomial time, that means, if there is some polynomial $P$ such that the algorithm runs in time at most $P(n)$ on

an input of length $n$ [6].

Class **P** is the set of decision problems that are solvable in polynomial time.

Class **NP** (Non-deterministic polynomial time) is a set of all problems for which there exists an efficient certifier. We call an algorithm $\mathcal{A}$ is an 'efficient certifier' for a problem $X$ if the following properties hold.

1. $\mathcal{A}$ is a polynomial time algorithm that takes two input arguments $s$ and $t$.
2. There is a polynomial function $P$ so that for every string $s$, we have $s \in X$ if and only if there exists a string $t$ such that $|t| \leq P(|s|)$ and $\mathcal{A}(s,t)$ =yes.

The relation between these complexity classes is that $\mathbf{P} \subseteq \mathbf{NP}$.

The famous unsolved '**P** versus **NP**' problem from computational complexity theory asks if $\mathbf{P} = \mathbf{NP}$? Many different attempts were made by researchers to solve this problem through which they built the theory of computations more wide and strong. For example, they explore approximate algorithms, parameterized algorithms in order to tackle with computationally intractability of the problems. We will explore about parameterized algorithms thoroughly in the next chapter.

### 2.3.1 NP-completeness

The class NP-complete contains computationally intractable problems for which there does not exist any efficient algorithm to solve them, at the same time we cannot prove that there will never be such an algorithm which solves these problems efficiently.

**Definition 8.** *[8]* ***(Polynomial-Time Reductions)*** *Let $A$ and $B$ be two decision problems. We say $A$ reduces (polynomial-time reduces) to $B$, denoted by $A \leq_P B$, if there exists a polynomial-time computable function $f$ such that $x$ is a yes-instance of $A$ if and only if $f(x)$ is a yes-instance of $B$.*

If there is a polynomial-time reduction from a decision problem $A$ to a decision problem $B$, then we say problem $B$ is at least as hard as problem $A$. If a decision problem $A$ is

polynomial-time reducible to a decision problem $B$ and $A$ is NP-hard then $B$ is NP-hard as well.

**Definition 9.** *[6]* ***(NP-hard)*** *A problem $A$ is said to be NP-hard if for every problem $B \in NP$, we have $B \leq_P A$.*

**Definition 10.** ***(NP-complete)*** *A problem is said to be NP-complete if it belongs to the class NP and it is NP-hard.*

**Lemma 1.** *[8] Suppose $X$ is an NP-complete problem. Then $X$ is solvable in polynomial time if and only if $\boldsymbol{P} = \boldsymbol{NP}$.*

**Lemma 2.** *[8] If $Y$ is an NP-complete problem and $X$ is a problem in NP with the property that $Y \leq_P X$, then $X$ is NP-complete.*

# Chapter 3

# Parametrized Complexity

In this chapter, we study an area of computational complexity known as parameterized complexity. Here, starting with the definition of parameterized problems, we further discuss parameterized algorithms.

## 3.1 Parameterized Problems and Parameterized Algorithms

Classical computational complexity theory targets towards finding algorithms to solve a problem which runs in time polynomial in the input instance size. So, the size of an input instance (n) is the only factor which is concerned. An input instance of a problem has more structure associated with it. There could be several factors apart from the size of input instance with respect to which we can try to find better algorithms than brute force.

Parameterized complexity is a field which exploits the structural properties of instances of a problem and classifies the problems into even finer classes than the classical computational complexity theory does.

We define a parameterized problem as follows:

**Definition 11.** *[1]* ***(Parameterized Problem)*** *A parameterized problem is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where $\Sigma$ is a fixed, finite alphabet. For an instance $(x, k) \in \Sigma^* \times \mathbb{N}$, $k$ is called*

*the parameter.*

Parameterized complexity aims towards finding algorithms to solve a problem whose running time depends on some computable function $f$ of parameter and a polynomial factor in $n$, where $n$ is the input size. Depending on the running time of the algorithms, the following parameterized complexity classes are defined.

**Definition 12.** *[1]* ***(Fixed Parameter Tractable)*** *A parameterized problem $L \subseteq \Sigma^* \times \mathbb{N}$ is called fixed-parameter tractable (FPT) if there exists an algorithm $\mathcal{A}$ (called a fixed-parameter algorithm), a computable function $f : \mathbb{N} \to \mathbb{N}$, and a constant $c$ such that, given $(x, k) \in \Sigma^* \times \mathbb{N}$, the algorithm $\mathcal{A}$ correctly decides whether $(x, k) \in L$ in time bounded by $f(k) \cdot |(x, k)|^c$. The complexity class containing all fixed-parameter tractable problems is called FPT.*

**Definition 13.** *[1]* ***(XP)*** *A parametrized problem $L \subseteq \Sigma^* \times \mathbb{N}$ is called slice-wise polynomial(XP) is there exists an algorithm $\mathcal{A}$ and two computable functions $f, g : \mathbb{N} \to \mathbb{N}$ such that, given $(x, k) \in \Sigma^* \times \mathbb{N}$, the algorithm $\mathcal{A}$ correctly decides whether $(x, k) \in L$ in time bounded by $f(k) \cdot |(x, k)|^{g(k)}$. The complexity class containing all slice-wise polynomial problems is called XP.*

## 3.2 Kernelization

Kernelization (Preprocessing or data reduction) is a useful technique used in parameterized complexity. It transforms a larger instance of a problem into its equivalent smaller instance. Two instances of a problem $Q$, $(x, k)$ and $(x', k')$ are said to be equivalent if $(x, k) \in Q$ if and only if $(x', k') \in Q$. Kernelization algorithm solves the easy part of the instance and reduces it into its core where the actual difficulty of the problem lies. It guarantees that larger instances of a particular parameterized problem can always be shrunk into smaller instances.

A kernelization algorithm consists of a series of safe Reduction Rules. Let us first define what a Reduction Rule is.

**Definition 14.** *[7]* ***(Reduction rule)*** *A data Reduction Rule or simply reduction rule, for a parameterized problem $Q$ is a function $\phi : \Sigma^* \times \mathbb{N} \to \Sigma^* \times \mathbb{N}$ that maps an instance $(I, k)$ of $Q$ to an equivalent instance $(I', k')$ of $Q$ such that $\phi$ is computable in time polynomial in $|I|$ and $k$.*

A reduction rule is said to be safe if the input instance and the reduced instance are equivalent instances of the problem.

**Definition 15.** *[7]* ***(Kernelization, Kernel)*** *Let $L$ be a parameterized problem over a finite alphabet $\Sigma$. A kernelization algorithm, or in short, a kernelization, for $L$ is an algorithm with the following property. For any given $(x, k) \in \Sigma^* \times \mathbb{N}$, it outputs in time polynomial in $|(x, k)|$, a string $x' \in \Sigma^*$ and an integer $k' \in \mathbb{N}$ such that*

$$((x, k) \in L \Leftrightarrow (x', k') \in L) \text{ and } |x'|, k' \leq h(k),$$

*where $h$ is an arbitrary computable function. If $K$ is a kernelization for $L$, then for every instance (x,k) of $L$, the result of running $K$ on the input (x,k) is called the kernel of (x,k) (under $K$). The function $h$ is referred to as the size of the kernel. If $h$ is a polynomial function, then we say that the kernel is polynomial.*

**Lemma 3.** *[7] A parameterized problem $P$ is FPT if and only if it admits a kernelization algorithm.*

## 3.3 W-Hierarchy

As mentioned earlier, parameterized complexity divides parameterized problems into fine classes. Here, we look into the most important parameterized complexity classes that come under W-hierarchy.

To define W-hierarchy, we need to define 'Parameterized Reductions' and 'Weighted Circuit Satisfiability problem'.

**Definition 16.** *[1]* ***(Parameterized Reductions)*** *Let $A, B \subseteq \Sigma^* \times \mathbb{N}$ be two parameterized problems. A parameterized reduction from $A$ to $B$ is an algorithm that, given an instance $(x, k)$ of $A$, outputs an instance $(x', k')$ of $B$ such that*

1. *$(x, k)$ is a yes-instance of $A$ if and only if $(x', k')$ is a yes-instance of $B$.*
2. *$k' \leq g(k)$ for some computable function $g$.*
3. *The running time is $f(k).|x|^{\mathcal{O}(1)}$ for some computable function $f$.*

**Lemma 4.** *[1] If there is a parameterized reduction from $A$ to $B$ and $B$ is FPT, then $A$ is FPT as well.*

| Weighted Circuit Satisfiability (WCS) |
|---|
| **Input:** A Boolean circuit $C$ and an integer $k$. |
| **Task:** Decide whether $C$ has a satisfying assignment of weight exactly $k$. |

A Boolean circuit is a directed acyclic graph with some input vertices and only one output vertex. In a circuit, if we define large nodes to be those with in-degree $> 2$ and small vertices to be those with in-degree $\leq 2$, then weft and depth of the circuit are defined as follows:

1. The weft of the circuit is the maximum number of large nodes appear on a path from an input vertex to output vertex. It is denoted by letter $t$.

2. The depth of the circuit is the length of the longest path from an input vertex to the output vertex.

$\mathcal{C}_{\mathrm{t,d}}$ denotes the class of circuits with weft at most $t$ and depth at most $d$.

Now, we are ready to define W-hierarchy.

**Definition 17.** *[1] **(W-hierarchy)** For $t \geq 1$, a parameterized problem $P$ belongs to the class $W[t]$ if there is a parameterized reduction from $P$ to $WCS[\mathcal{C}_{t,d}]$ for some $d \geq 1$.*

The inclusion wise relations among all the classes mentioned so far in this chapter is as follows:

$$FPT \subseteq W[1] \subseteq W[2] \subseteq W[3] \subseteq \cdots \subseteq W[t] \subseteq \cdots \subseteq XP.$$

# Chapter 4

# Parameterized Complexity of Fair Feedback Vertex Set Problem

In this chapter, we show that Fair Feedback Vertex Set (FFVS) problem defined in section 1.1 is $W[1]$-hard with respect to some structural graph parameters like treewidth and treedepth of the graph. Here the instances of simple graphs and multigraphs are dealt separately.

## 4.1 On multigraphs, W[1]-hard with respect to treewidth

In order to prove $W[1]$-hardness results, we perform parameterized reductions from Fair Vertex Cover problem (Fair VC) to FFVS problem. Knop, Masarík and Toufar obtained a reduction from Multicolored Clique problem to Fair VC problem and proved that Fair VC problem is $W[1]$-hard with respect to parameter $fvs(G) + td(G)$ [3].

| Fair Vertex Cover Problem (Fair VC) |
| --- |
| **Input:** A graph $G = (V, E)$ and a positive integer $\ell$. |
| **Task:** To decide whether there exists a set $S \subseteq V(G)$ such that $S$ covers all edges in the graph $G$ and $\max_{v \in V(G)} \{\lvert N(v) \cap S \rvert\} \leq \ell$. |

We say a set $S$ covers all edges of graph $G$ if every edge in $G$ is incident on at least one of the vertices from the set $S$. For such set the induced graph $G[V \setminus S]$ is edgeless. In a similar way, a subset $S$ of vertex set of a graph covers all cycles in the graph, if $S$ contains at least one vertex from each of the cycle. Here, graph $G \setminus S$ is a forest.

**Theorem 5.** *FFVS on multigraph is W[1]-hard with respect to the treewidth of the multigraph.*

**Proof.** We obtain a parameterized reduction from Fair VC problem, which is W[1]-hard with respect to parameters $(fvs(G) + td(G))$, to FFVS problem. We construct an instance $(H, \ell')$ of FFVS problem starting with an instance $(G, \ell)$ of Fair VC problem as follows: Given an instance $(G, \ell)$, let $V(H) = V(G)$, and for every edge in $G$, we put two parallel edges between the corresponding endpoints in $H$. let $\ell' = \ell$. The new instance can be obtained in polynomial time in the input instant's size. See Figure 4.1.

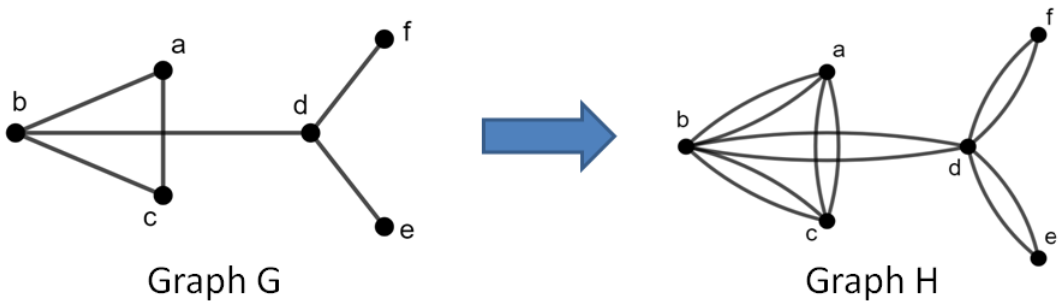



Figure 4.1: Construction of an input instance from Fair VC problem to FFVS problem

Now, we claim that $(G, \ell)$ is a yes-instance of Fair VC problem if and only if $(H, \ell)$ is a yes-instance of FFVS problem.

For the forward implication, if $(G, \ell)$ is a yes-instance of Fair VC problem then there exists a set $X \subseteq V(G)$ such that $V(G) \setminus X$ is an independent set and $\max_{v \in V(G)}\{|N(v) \cap X|\} \leq \ell$. We prove that the same set $X \subseteq V(H)$ is a solution of instance $(H, \ell)$ of FFVS problem. Since $X$ covers all edges in graph $G$, it also covers all edges in graph $H$ because we only had added edges between vertices in $H$ which were adjacent in $G$. So, $V(H) \setminus X$ is an independent set and so does a forest. Moreover, the fair cost of the set $X$ in $H$ is also less than $\ell$. Hence, $(H, \ell)$ is a yes-instance of FFVS problem.

Conversely, if $(H, \ell)$ is a yes-instance of FFVS problem, then there exists a set $Y \subseteq V(H)$ such that $H \setminus Y$ is a forest and $\max_{v \in V(H)}\{|N(v) \cap Y|\} \leq \ell$. Since $Y$ covers all cycles in the graph $H$, it also covers all cycles formed by parallel edges introduced in the construction. As a result, $Y$ covers all edges in $H$. Therefore, the corresponding set $Y \subseteq V(G)$ covers all edges in the graph $G$. Fair cost of the set $Y$ in $G$ does not exceed $\ell$, resulting, $Y \subseteq V(G)$ to be a solution of Fair VC problem. Thus, $(G, \ell)$ is a yes-instance of Fair VC problem.

Here, the parameter used for FFVS problem is treewidth of the multigraph. To check the second condition of the parameterized reduction (definition 16), we need to show the boundedness between the parameters. It can be shown by proving following inequality for some computable functions $f_1$ and $f_2$.

$tw(H) \leq g(td(G) + fvs(G))$

That is, to show, $tw(H) \leq f_1(td(G)) + f_2(fvs(G))$

Proving the above inequality is equivalent to proving, $tw(G) \leq f_1(td(G)) + f_2(fvs(G))$, as $tw(H) = tw(G)$ is true, since every tree decomposition of graph $G$ is also a tree decomposition of graph $H$.

Thus, it suffices to prove the following two inequalities:

(1) $tw(G) \leq f_1(td(G))$ and

(2) $tw(G) \leq f_2(fvs(G))$.

For (1), if $td(G)$ is an integer $a$, then there exists a rooted forest $F$, of depth $a$, on the vertices of $G$ such that its transitive closure contains graph $G$. We put all vertices along a path from root $r$ to a leaf in a bag and connect all the bags accordingly to obtain a valid tree decomposition of $H$. The width of this tree decomposition would be at most $a-1$. This would imply, $tw(G) \leq (a-1)$ and hence $tw(G) \leq td(G) - 1$. We illustrate this idea through an example in Figure 4.2.

For (2), if $fvs(G)$ is an integer $b$, there exists a set $X \subseteq V(G)$ with $|X| = b$ such that $G \setminus X$ is a forest. For graph $G \setminus X$, a tree decomposition $T$ of width at most one can be obtained. We add $X$ to each of the bags of this tree decomposition $T$ and obtain a tree decomposition of graph $G$ of width at most $1 + b$. Hence, $tw(G) \leq 1 + b = 1 + fvs(G)$. We

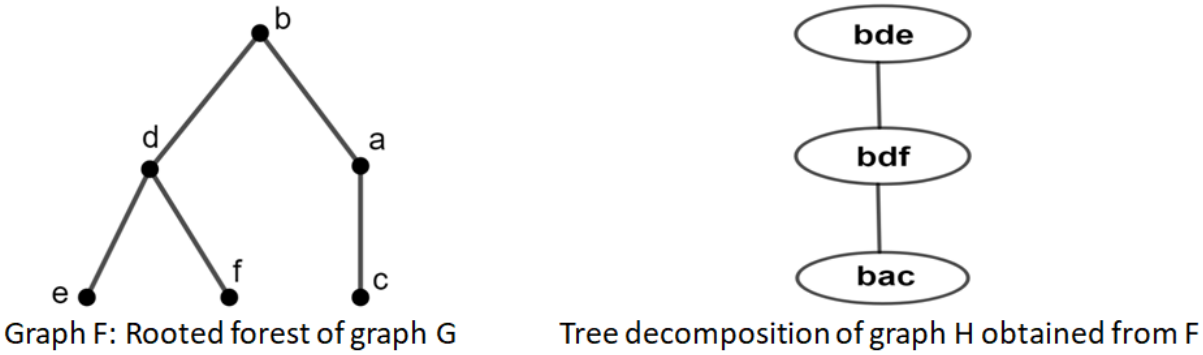



Figure 4.2: Boundedness of $tw(H)$ by $td(G)$

illustrate this idea with an example in Figure 4.3. Consider the graph $G$ shown in Figure 4.1. Let $X = \{b\}$. Graph $G \setminus X$, tree decomposition of $G \setminus X$ and finally tree decomposition of $G$ are shown in Figure 4.3.

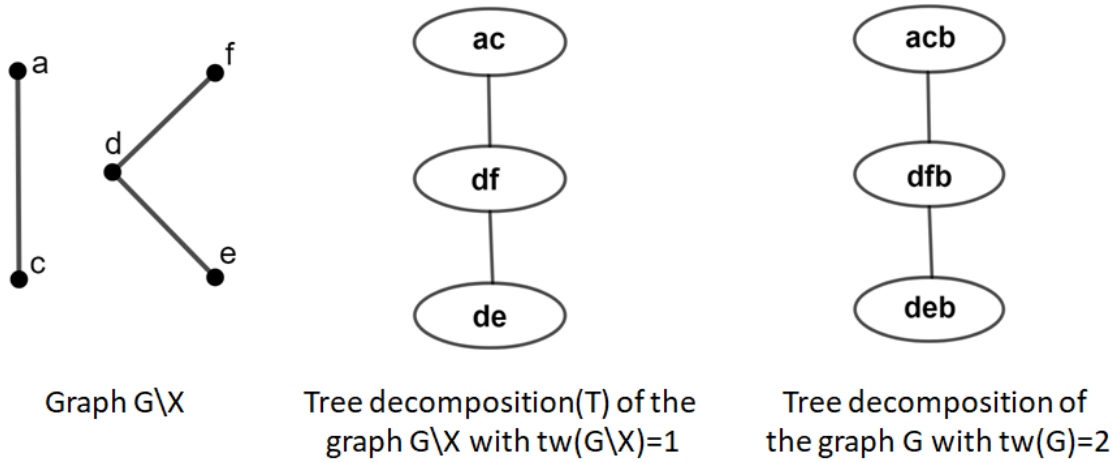



Figure 4.3: Boundedness of $tw(H)$ by $fvs(G)$

Thus, the parameter $tw(H)$ of FFVS problem is bounded by the parameter $td(G)+fvs(G)$ of Fair VC problem.

## 4.2 FFVS problem on simple graphs is W[1]-hard with respect to treedepth

**Theorem 6.** *FFVS problem on simple graphs is $W[1]$-hard with respect to treedepth of the graph.*

**Proof.** Similar to the previous theorem, here, we aim to construct an instance of FFVS problem from an instance of Fair VC problem, where the constructed instance of FFVS is a simple graph. Let $(G, \ell)$ be an instance of Fair VC problem. Then an instance $(G', \ell')$ of FFVS problem is constructed as follows (See Figure 4.4).

Graph $G'$ contains all the vertices of $G$ along with their edges in $G$, we call these vertices as 'original vertices' and edges as 'original graph edges'. $G'$ also contains $4\ell + 2$ new vertices $\{e(a_1), e(b_1), e(a_2), e(b_2), \ldots, e(a_{2\ell+1}), e(b_{2\ell+1})\}$ for each edge $e \in E(G)$. For each edge $e = uv$, we denote this set of new vertices by $E_{uv}$. We make $e(a_i)$ adjacent to $u$ and $e(b_i)$ adjacent to $v$ and $(e(a_i), e(b_i)) \in E(G')$ for all $1 \leq i \leq 2\ell+1$. This construction results in a formation of a 4-cycle between vertices $u, v, e(a_i), e(b_i)$. Finally we set $\ell' = \ell$. This new instance can be constructed in polynomial time in the size of an input instance.

We need to show that $(G, \ell)$ is a yes-instance of Fair VC problem if and only if $(G', \ell)$ is a yes-instance of FFVS problem.

Suppose $(G, \ell)$ is a yes-instance of Fair VC problem, that means, there exists a set $X \subseteq V(G)$ such that $G \setminus X$ is an independent set and $\max_{v \in V(G)}\{|N(v) \cap X|\} \leq \ell$. Then we claim that the same set $X$, when deleted from $G'$ results in a forest. Note that, every cycle in $G'$ contains at least one original graph $G$ edge. As set $X$ covers all edges in $G$, it covers all original graph edges in $G'$. Therefore set $X$ covers all cycles in $G'$. That is why, $G' \setminus X$ results in a forest. Also, the newly added vertices can have at most one neighbour in the solution set $X$. Hence, as $\ell \geq 1$, new vertices do not affect the fair cost of the solution set. Hence $X \subseteq V(G')$ is a solution to the instance $(G', \ell)$ of FFVS problem. Thus, $(G', \ell)$ is a yes-instance of FFVS problem.

For the reverse direction, suppose $(G', \ell)$ is a yes-instance of FFVS problem. Then there exists a set $Y \subseteq V(G')$ such that $G' \setminus Y$ is acyclic and $\max_{v \in V(G')}\{|N(v) \cap Y|\} \leq \ell$. Now we prove that $Y$ contains at least one endpoint of each edge $(u, v)$, where $(u, v)$ is an original

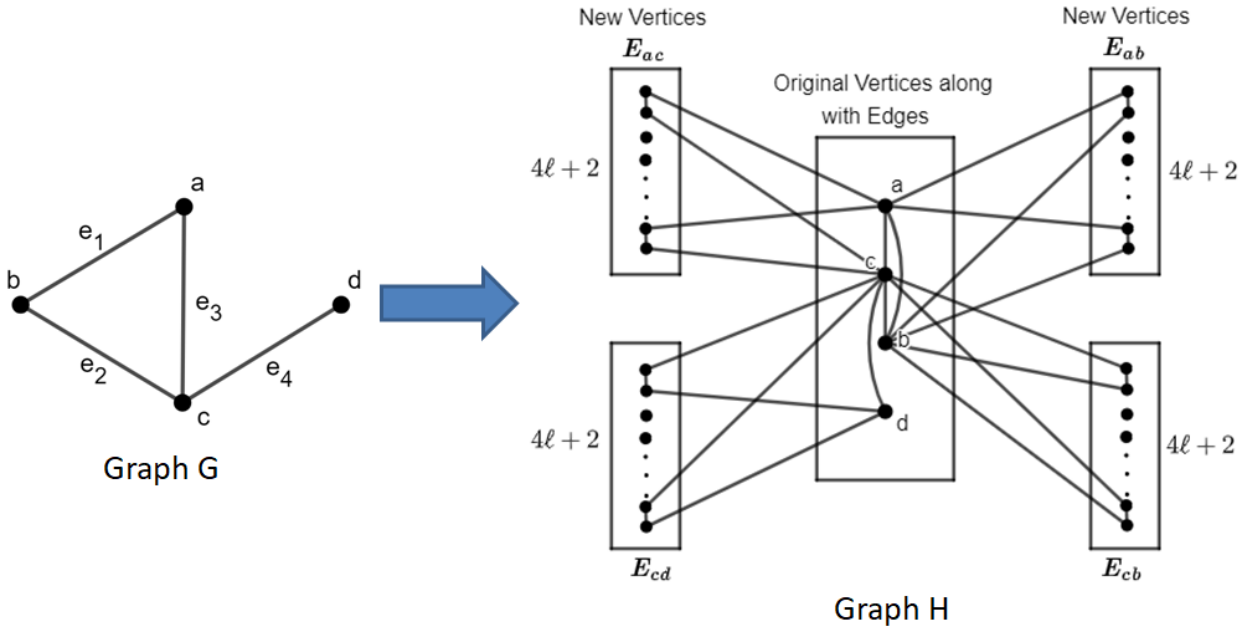



Figure 4.4: Construction of an input instance of FFVS problem from Fair VC problem

graph edge. Suppose not, then for an original graph edge $(u, v)$, we can delete at most $\ell$ vertices adjacent to each of $u$ and $v$. This helps us to eliminate at most $2\ell$ cycles. According to the construction, there is still one more cycle left. This contradicts that $Y$ is a fair feedback vertex set of $G'$. Hence for each original edge $e$ in $G'$, the set $Y$ has to contain at least one end-point of $e$ in a solution set of the FFVS problem. This implies that Y covers all original edges in $G'$. Set $S = Y \cap V(G)$. Note that, $S$ serves as a solution for $(G, \ell)$ of Fair VC problem. The fair cost of set $S$ in $G'$ is already less than $\ell$, so the fair cost of $S$ in $G$ is also less than $\ell$. Hence, $(G, \ell)$ is a yes-instance of the Fair VC problem.

Now, we are only left to show how parameter $td(G')$ of FFVS problem is bounded by parameter $fvs(G) + td(G)$ of Fair VC problem.

Let $td(G)$ be equal to $a$. Then there exists a rooted forest $F$ of depth $a$ such that its transitive closure contains graph $G$. Using the rooted forest $F$ mentioned above, a rooted forest $F'$ of depth at most $a+2$ can be constructed such that transitive closure of $F'$ contains graph $G'$. This would imply $td(G') \leq a + 2 \leq td(G) + 2 \leq fvs(G) + td(G) + 2$. To prove this result, it is enough to construct such a forest $F'$ using $F$. Note that, given an edge $e = (u, v) \in E(G)$, there exists a path $P$ joining a root and a leaf node $w$ in $F$ such that both $u$ and $v$ lie on $P$. That means, $u$ and $v$ have ancestor-descendant relationship to each

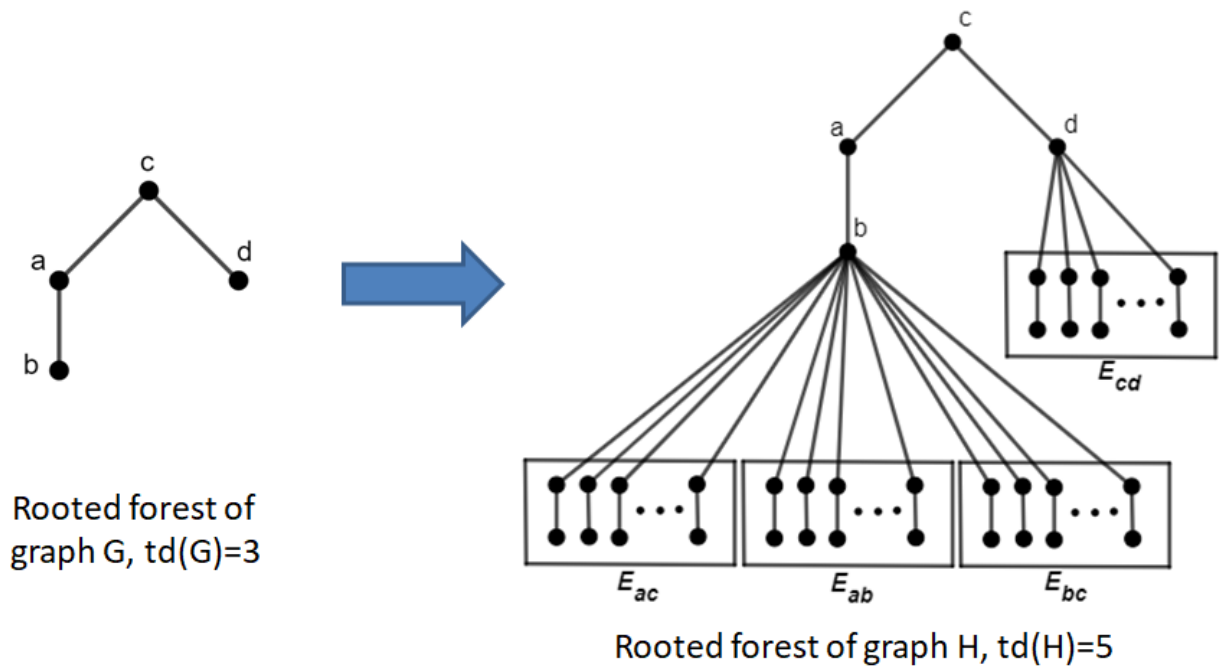



Figure 4.5: Boundedness of $td(G')$ by $td(G)$

other in $F$. Find such a path $P$ and select $w$. As shown in Figure 4.5, forest $F$ can be further extended by adding $2\ell+1$ branches to the leaf $w$, where $i^{th}$ branch contains two new vertices $e(a_i)$ and $e(b_i)$ one after another. This is repeated for all edges to obtain the required forest $F'$.

## 4.3 FFVS on simple graphs is W[1]-hard with respect to treewidth

**Theorem 7.** *FFVS on simple graph is W[1]-hard when parameterized by treewidth of the graph.*

**Proof.** To prove this theorem, we follow the same construction used in Theorem 6. Previously, starting with an instance $(G, \ell)$ of Fair VC problem, we built an instance $(G', \ell)$ of FFVS problem. We also proved that $(G', \ell)$ is a yes-instance of FFVS problem iff $(G, \ell)$ is a yes-instance of Fair VC problem. In order to complete the proof of this theorem, we only

need to show that the parameter $tw(G')$ of FFVS problem is bounded by the parameter $td(G) + fvs(G)$ of Fair VC problem.

In the proof of Theorem 6, we constructed a rooted forest $F'$ such that its transitive closure contains graph $G'$. We also proved that depth of this forest $F'$ is bounded above by $td(G) + 2$. By tracing the forest $F'$, it is possible to construct a tree decomposition of graph $G'$. Corresponding to each leaf $w$ in $F'$, there is a path from root $r'$ of $F'$ to $w$. We put all vertices on this path from $r'$ to $w$ into one bag and continue doing this for all the leaves of $F'$. Now we have the collection of bags required to form a tree decomposition. It is easy to make these bags adjacent so that they form a valid tree decomposition. We obtain a tree decomposition of graph $G'$ using the rooted forest $F'$ and illustrate this in Figure 4.6. Observe that, for all edges $e$ in $G'$, at least one path from $r'$ to a leaf of the forest $F'$ contains both the endpoints of the edge $e$. Hence all conditions of the tree decomposition are satisfied. Thus, the width of the tree decomposition so formed is at most $td(G') - 1$. Hence $tw(G') \leq td(G') - 1 \leq td(G) + 2 - 1 \leq td(G) + 1 \leq fvs(G) + td(G) + 1$.

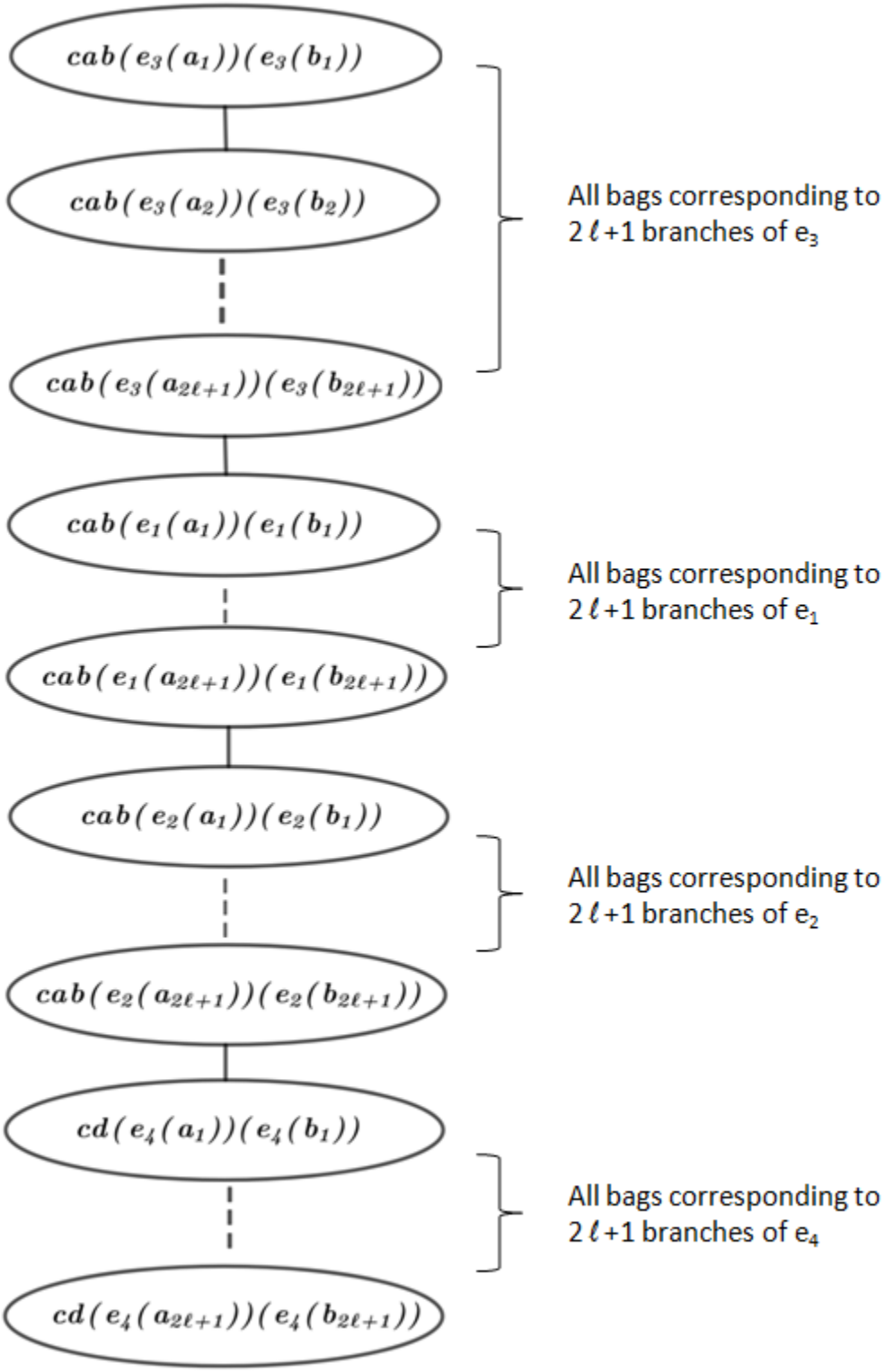



Figure 4.6: Tree decomposition of graph $H$ shown in Figure 4.4

# Chapter 5

# An FPT algorithm for FFVS Problem

## 5.1 FPT with respect to the neighbourhood diversity

Till now, we showed $W[1]$-hardness of FFVS problem with respect to some structural graph parameters, mainly treedepth and treewidth of the graph. In this section, we obtain an FPT algorithm to solve the problem with respect to a structural graph parameter called neighbourhood diversity, denoted by $nd(G)$.

The main idea behind the algorithm is based on a few observations. These observations help in order to decrease the total number of searches through all possible feedback vertex sets of the graph. We state the observations below:

**Observation 8.** *Consider a graph $G$ and a Feedback Vertex Set $X$ of $G$. If $u$ and $v$ are twin vertices and $u \in X$ whereas $v \notin X$, then we can replace $u$ by $v$ in set $X$ and the resultant set $X' = (X \setminus \{u\}) \cup \{v\}$ is another FVS of $G$. Two vertices $u, v \in V(G)$ are said to be twin vertices if they satisfy the criteria $N(u) \setminus \{v\} = N(v) \setminus \{u\}$.*

We say such Feedback vertex sets $X$ and $X'$ are of the 'same type' if one set can be obtained from another by replacing a few of the vertices by their twin vertices.

**Observation 9.** *If $X$ and $X'$ are two distinct FVS of graph $G$ of the same type, then*

$$\max_{v \in V(G)} \{|N(v) \cap X|\} = \max_{v \in V(G)} \{|N(v) \cap X'|\}$$

For any two feedback vertex sets $X$ and $X'$ of the same type, the fair cost of the sets in the graph remains the same.

**Theorem 10.** *There exists an FPT algorithm running in time $3^k k^2 n^{\mathcal{O}(1)}$ for FFVS problem, where $k$ is the neighbourhood diversity of the graph.*

**Proof.** If neighbourhood diversity of a graph is bounded by an integer $k$, then there exists a partition $P$ of $V(G)$ into $k$ classes, $P = \{V_1, V_2, V_3, ..., V_k\}$, such that all vertices in one class have the same neighbourhood, that is, $N(u) \setminus \{v\} = N(v) \setminus \{u\}$, if $u, v \in V_i$.

We observe the following facts about the partition $P$:

- Each class $V_i$ could either be a clique or an independent set.
- We say classes $V_i$ and $V_j$ are adjacent when every vertex $u \in V_i$ is adjacent to every vertex $v \in V_j$.
- We say classes $V_i$ and $V_j$ are nonadjacent when there is no edge $(u, v)$ such that $u \in V_i$ and $v \in V_j$.
- Two classes in partition $P$ can either be adjacent or non-adjacent.

Let us define a graph $Q$ called partition graph, corresponding to a partition $P = \{V_1, V_2, V_3, ..., V_k\}$.
Set $V(Q) = \{v_1, v_2, v_3, ..., v_k\}$, where vertex $v_i$ corresponds to class $V_i$ and
Set $E(Q) = \{(v_i v_j) |$ $V_i$ and $V_j$ are adjacent in the partition $P.\}$

Let $A(i, j)$ be defined by,

$$A(i,j) = \begin{cases} 1, & \text{if } v_i v_j \in E(Q) \\ 0 & \text{otherwise} \end{cases}$$

From observations 8 and 9, since the same type of feedback vertex sets have the same fair cost, it suffices to check the fair cost of all different types of feedback vertex sets than

checking over all possible feedback vertex sets. Given a partition $P = \{V_1, V_2, V_3, ..., V_k\}$ of $V(G)$, where $k$ is neighbourhood diversity of $G$, we construct different types of feedback vertex sets as follows:

(a) If class $V_i$ is a clique and $X$ is an FVS of graph $G$, then $X$ contains either
1. all vertices of class $V_i$,
2. all but one vertex from class $V_i$, or
3. all but two vertices from class $V_i$.

(b) If class $V_i$ is an independent set and $X$ is an FVS of graph G, then $X$ contains either
1. all vertices of class $V_i$,
2. all but one vertex from class $V_i$, or
3. no vertex from class $V_i$.

The validity and the sufficiency of all the above cases can be explained. There is no other case possible for cliques apart from those mentioned in $(a)$. When $V_i$ is a clique, if we do not consider three vertices in $X$, clearly these three vertices form a triangle. Therefore, $X$ is not an FVS.

Now, when $V_i$ is an independent set, we will show that it is sufficient to consider the three cases given in $(b)$. Apart from the cases mentioned in $(b)$, suppose $X$ is an FVS that contains all vertices of $V_i$ except two vertices $u, v \in V_i$. As $X$ is an FVS, $u$ and $v$ do not form any cycle in $G \setminus X$. Let $X' = X \setminus V_i$. Since all the vertices of $V_i$ have the same neighbourhood as $u$ and $v$, none of the vertex in $V_i$ is part of any cycle in $G \setminus X'$. Hence, $G \setminus X'$ is also acyclic. Thus $X$ contains all but two vertices from $V_i$ implies $X'$ contains no vertex from $V_i$, which is covered in $b - 3$.

Thus, based on whether a class is a clique or an independent set, we choose one option out of three from $(a)$ or $(b)$. Then we combine these to get a set which could potentially be a FVS of the graph.

Combinations generated by all the above cases in $(a)$ and $(b)$ give us all possible FVS up to equivalence by the 'same type' relation. According to the observation 2, it does not matter which vertices are being chosen from a class as all vertices in a class are twin vertices. Hence a different combination of selection of vertices produces feedback vertex set of the same type.

Given a partition of the vertex set $V(G)$ into $k$ classes according to the neighbourhood diversity, $3^k$ possible subsets $X \subseteq V(G)$ can be built as per the criteria mentioned in (a) and (b) above. Note that not each of $3^k$ choices produces an FVS. Now we aim to check, if a selected subset indeed gives us an FVS of the graph and subsequently check the fair cost of that set.

Once we select a combination for the formation of set $X$ from those $3^k$ possible combinations, we know exactly how many vertices from a particular class have been selected in the set $X$. Let, $[a_i]$ be the number of vertices selected from class $V_i$ in $X$.

We can check, if a set $X$ is an FVS of graph $G$ in time $O(n)$. We proceed further, only if, $X$ is an FVS. After that, we are only left to check if the set $X$ satisfies the criteria, $\max_{v \in V(G)}\{|N(v) \cap X|\} \leq \ell$, which can also be checked by verifying the following conditions:

When $V_i$ is an independent set, verify if

$$\sum_{j=1, j \neq i}^{k} A(i,j)[a_j] \leq \ell.$$

When $V_i$ is a clique, verify if

$$[a_i] - 1 + \sum_{j=1, j \neq i}^{k} A(i,j)[a_j] \leq \ell.$$

If for at least one $X$, above conditions are true for all $k$ classes, then the given instance is a yes-instance of FFVS problem. Otherwise, it is a no-instance.

Calculating function $A(i,j)$ takes $\mathcal{O}(k^2)$ time. We have total $3^k$ possible combinations to check. It takes $\mathcal{O}(n)$ time to check if a set $X$ obtained by a combination is an FVS of graph $G$. Calculating values of $[a_i]$ for a combination takes $\mathcal{O}(k)$ times. Verifying $k$ conditions corresponding to $k$ classes takes $\mathcal{O}(k^2)$ time. Hence, the time complexity of this algorithm is $3^k k^2 n^{\mathcal{O}(1)}$.

# Chapter 6

# Min-FFVS Problem

## 6.1 Min-FFVS problem is NP-complete

In this section, we prove that the Minimum Fair Feedback Vertex Set problem (Min-FFVS) is NP-complete. The reduction that we obtain here is from the Feedback Vertex Set problem (FVS). Feedback Vertex Set problem is among the first few problems which were shown to be NP-Complete in the classical complexity theory. We first define the Min-FFVS problem.

| Minimum Fair Feedback Vertex Set (Min-FFVS) |
|---|
| **Input:** A simple undirected graph $G$ and positive integers $\ell$ and $k$. |
| **Task:** To check whether there exists a set $S \subseteq V(G)$, $\lvert S\rvert \leq k$ such that $G \setminus S$ is an acyclic graph and $\max_{v \in V(G)} \{\lvert N(v) \cap S\rvert\} \leq \ell$. |

**Theorem 11.** *Min-FFVS problem is NP-complete.*

**Proof.** First, we prove that Min-FFVS problem is in NP. A certificate could be a subset $X \subseteq V(G)$, and a certifier could then check if $X$ is indeed an FVS of size at most $k$ and $|N(v) \cap X| \leq \ell$ for all $v \in V(G)$. A certifier could compute $G \setminus X$ and run DFS on $G \setminus X$ to identify a non-tree edge. If there is no non-tree edge, then $G \setminus X$ is a forest. This takes $\mathcal{O}(V + E)$ times. For more details refer [11]. We also need to make sure that there are not more than $\ell$ vertices in $X$ from the neighbourhood of any vertex in the graph. This can be

checked by calculating $|N(v) \cap X|$ for all vertices in poly time.

Next, we prove that FVS $\leq_P$ Min-FFVS, which shows that min-FFVS is NP-hard. The reduction algorithm takes as input an instance $(G, k)$ of FVS problem. The output of the reduction algorithm is the instance $(G, k, k)$ of Min-FFVS problem.

We claim that $(G, k)$ is a yes-instance of FVS problem if and only if $(G, k, k)$ is a yes-instance of Min-FFVS problem. Suppose $G$ has a feedback vertex set $X \subseteq V$ with $|X| \leq k$. Now, since $|X| \leq k$, $|N(v) \cap X| \leq k$ for all $v \in V(G)$. Hence the solution set $X$ is fair. Which implies $(G, k, k)$ is a yes-instance of Min-FFVS problem. Conversely, Suppose $G$ has a Min Fair Feedback Vertex Set $Y \subseteq V$ with $|Y| \leq k$ and $|N(v) \cap Y| \leq k$ for all $v \in V$. The same set $Y$ is also FVS of size at most $k$ in $G$. Thus $(G, k)$ is a yes-instance of FVS problem. This proves that Min-FFVS is NP-hard.

## 6.2 Kernel for Min-FFVS using parameter $\Delta + k$

We find the kernelization of Min-FFVS with respect to parameter $\Delta + k$, where $\Delta$ is the maximum degree in the graph and $k$ is the size of fair feedback vertex set.

Note that, we can't delete any vertex because we have to keep a track of how many vertices are going in the Min-FFVS from a neighbourhood of a vertex. We start by enlisting a couple of reduction rules to reduce the instance of a Min-FFVS problem.

**Reduction Rule 1.** *Given an instance $(G, k, \ell)$, if there exists a vertex $v \in V(G)$ such that $d(v) \leq 1$, then delete $v$, and reduce $(G, k, \ell)$ to $(G \setminus v, k, \ell)$.*

Reduction Rule 1 is safe. A vertex with degree at most 1 do not participate in any cycle of the graph. Thus, its removal does not change the solution.

**Reduction Rule 2.** *If there exists a sequence $\{v_1, v_2, \ldots, v_p\}$ with $p > 3$, $v_i \neq v_j$, $d(v_i) = 2$ and $(v_i, v_{i+1}) \in E$, then delete vertices $v_3, v_4, \ldots, v_{p-1}$ and introduce an edge between $v_2$ and $v_p$. In other words, we replace the sequence $\{v_1, \ldots, v_p\}$ by $\{v_1, v_2, v_p\}$. The reduced instance is $(G \setminus \{v_3, v_4, \ldots, v_{p-1}\}, k, \ell)$.*

Reduction Rule 2 is safe because every minimal FVS of a graph contains at most one element from a sequence of degree-2 vertices. If $(G, k, \ell)$ is a yes-instance, there is a FFVS $S$ of size at

most $k$. If $S$ contains a vertex from a long degree-2 sequence, then we replace it by the middle vertex $v_2$ from reduced degree-2 sequence of length three in the graph $G \setminus \{v_3, v_4, \ldots, v_{p-1}\}$ and obtain a solution of reduced instance. On the other hand, if $(G \setminus \{v_3, v_4, \ldots, v_{p-1}\}, k, \ell)$ is a yes-instance of Min-FFVS problem, then the solution $S'$ of $(G \setminus \{v_3, v_4, \ldots, v_{p-1}\}, k, \ell)$ also satisfies the instance $(G, k, \ell)$.

**Lemma 12.** *If $(G, k, \ell)$ is a yes-instance of Min-FFVS problem and none of the above reduction rules are applicable, then $|V(G)| \leq (k + 8\Delta k - 3)$.*

**Proof.** If $(G, k, \ell)$ is a yes-instance, then there exists a set $S \subseteq V(G)$, with $|S| \leq k$ and $F = G \setminus S$ is a forest and $|N(v) \cap S| \leq \ell$ for all $v \in V(G)$. See Figure 6.1. Then vertices of $F$ can be partitioned into four parts $\{D_{\leq 1}, D_{\geq 3}, D_{2(orig)}, D_{2(new)}\}$, where
$D_{\leq 1}$ = the set of vertices from $F$ such that $d_F(v) \leq 1$. $d_F(v)$ denotes degree of $v$ in $F$.
$D_{\geq 3}$ = the set of vertices from $F$ such that $d_F(v) \geq 3$.
$D_{2(new)}$ = the set of vertices from $F$ such that $d_F(v) = 2$ and $d_G(v) > 2$ in $G$.
$D_{2(orig)}$ = the set of vertices from $F$ such that $d_F(v) = 2$ and $d_G(v) = 2$ in $G$.

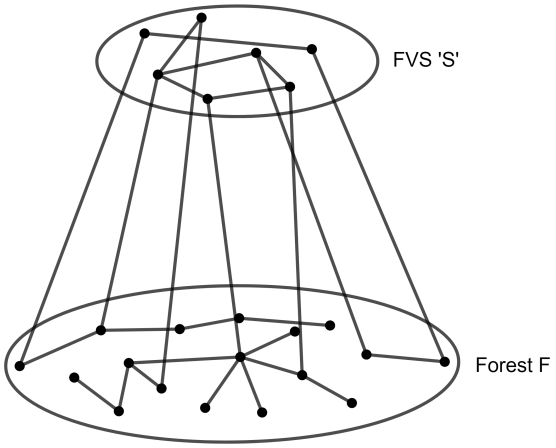



Figure 6.1: FVS $S$ and the remaining forest $F$

Set $S$ contains at most $k$ vertices and every vertex in $G$ has degree at most $\Delta$. This implies $|N(S)| \leq \Delta k$. A point to note is that $D_{\leq 1}$ and $D_{2(new)}$ must have a neighbour in $S$, as their degree in $F$ is reduced. Hence, $|D_{\leq 1} \cup D_{2(new)}| \leq \Delta k$.

As $F$ is a forest, we have $|D_{\geq 3}| \leq |D_{\leq 1}| \leq \Delta k$. Now, we claim that $|D_{2(orig)}| \leq 3(2\Delta k - 1)$. Since we cannot apply Reduction Rule 2, there are sequences $\{u_1, u_2, u_3\}$ of degree two vertices. $u_1$ and $u_3$ are adjacent to vertices in $D_{\leq 1} \cup D_{\geq 3} \cup D_{2(new)}$. We know $D_{\leq 1} \cup D_{\geq 3} \cup$

$D_{2(new)} \leq 2\Delta k$; these vertices can have at most $2\Delta k - 1$ edges among them without forming a cycle. Hence there can be at most $2\Delta k - 1$ sequences of size 3 of degree two vertices. Therefore, we have $|D_{2(orig)}| \leq 3(2\Delta k - 1)$. Thus, the total number of vertices in $G$ is at most $k + \Delta k + \Delta k + 3(2\Delta k - 1) = k + 8\Delta k - 3$.

**Reduction Rule 3.** *If the total number of vertices in the reduced graph is more than* $(k + 8\Delta k - 3)$*, then conclude that we are dealing with a no-instance.*

Reduction Rule 3 is safe because of Lemma 12. This gives us the following theorem.

**Theorem 13.** *Min-FFVS problem admits a kernel with* $\mathcal{O}((\Delta + k)^2)$ *vertices.*

# Chapter 7

# Relax-FFVS Problem

In Relax-FFVS problem, we have a weaker condition on the budgets of the vertices in the graph. In this problem, instead of demanding for all vertices to satisfy the budget, we only demand the vertices outside the fair feedback vertex set to satisfy the condition $|N(v) \cap S| \leq \ell$, where $S$ is a FFVS. Here, we begin with the definition of Relax-FFVS problem.

| Relaxed Fair Feedback Vertex Set (Relax-FFVS) |
| --- |
| **Input:** A Simple undirected graph $G$ and positive integers $\ell$ and $k$. |
| **Task:** Decide whether there exists a set $S \subseteq V(G)$ with $\|S\| \leq k$ such that $G \setminus S$ is a forest and for every vertex $v \in V(G) \setminus S$ it holds that $\|N(v) \cap S\| \leq \ell$. |

## 7.1 FPT when parameterized by solution size

In this section, we obtain an FPT algorithm for Relax-FFVS problem with respect to the parameter $k$. Given a graph $G$, if there does not exist a FVS of size $k$, then surely there is no relax fair feedback vertex set of size $k$.

We start with finding a feedback vertex set $S$ of size at most $k$ for an input instance $(G, k, \ell)$ of Relax-FFVS problem. The next step is to consider all partitions of $S$ into two parts. Given a partition $S = X \cup Y$, if $G[Y]$ is acyclic, then only solve the disjoint version of Relax-FFVS problem, which is defined as follows:

| Disjoint Relax-FFVS |
| --- |
| **Input:** A simple undirected graph $G$, a feedback vertex set $S$ of $G$, a partition of $S = X \cup Y$ and two positive integers $\ell$ and $k$. |
| **Task:** Does there exist a feedback vertex set $Y'$ of $G \setminus X$ of size at most $k - \|X\|$ such that $Y' \cap Y = \phi$ and $\|N(v) \cap (X \cup Y')\| \leq \ell$ for all $v \in G \setminus (X \cup Y')$. |

**Lemma 14.** *Disjoint Relax-FFVS is solvable in* $\mathcal{O}^*(16^k)$.

**Proof.** In the disjoint version of the problem, we fix the part $X$ of $S$ in the final solution and replace the part $Y$ by some set $Y'$ disjoint from $Y$ such that the required conditions are being satisfied. In this algorithm, instead of finding $Y'$ separately, we make changes in $X$ itself. So the original set $X$ when subtracted from the final updated set $X$ gives us $Y'$. Now, we introduce some Reduction Rules which make our task easier.

**Reduction Rule 4.** *For an instance* $(G, X, Y, k, \ell)$ *of Disjoint Relax-FFVS problem, if* $d_{G \setminus X}(v) \leq 1$ *for* $v \in V(G \setminus X)$*, delete* $v$*. The reduced instance is* $(G \setminus \{v\}, X, Y \setminus \{v\}, k, \ell)$.

Reduction Rule 4 is safe. Because a vertex with degree at most one cannot be a part of any cycle in the graph. After the application of this Reduction Rule, the minimum degree of the graph $G \setminus X$ is 2.

**Reduction Rule 5.** *If there is a path* $\langle u_1, v_1, v_2, \ldots, v_p, u_2 \rangle$ *in graph* $G \setminus X$ *with* $p > 3$*,* $d(v_i) = 2$*,* $d(u_i) > 2$*, then shrink this path* $\langle u_1, v_1, v_2, \ldots, v_p, u_2 \rangle$ *to* $\langle u_1, v_1, v_2, v_3, u_2 \rangle$*. The reduced instance is* $(G \setminus \{v_4, v_5, \ldots, v_p\}, X, Y \setminus \{v_4, v_5, \ldots, v_p\}, k, \ell)$*. Now, if* $v_i \in Y$ *for some* $i \geq 4$*, put* $v_1$ *in* $Y$*. The reduced instance in this case is* $(G \setminus \{v_4, v_5, \ldots, v_p\}, X, \{Y \setminus \{v_4, v_5, \ldots, v_p\} \cup \{v_1\}\}, k, \ell)$.

Reduction Rule 5 is safe. If $(G, X, Y, k, \ell)$ is a yes-instance and its solution $S$ contains $v_i, i \geq 4$, then the solution $S'$ of the reduced instance is $S' = (S \setminus v_i) \cup \{v_2\}$. Conversely, if $(G \setminus \{v_4, v_5, \ldots, v_p\}, X, Y, k, \ell)$ or $(G \setminus \{v_4, v_5, \ldots, v_p\}, X, \{Y \setminus \{v_4, v_5, \ldots, v_p\} \cup \{v_1\}\}, k, \ell)$ is a yes-instance and $S'$ is a solution of this instance, then the same $S'$ is a solution of instance $(G, X, Y, k, \ell)$.

**Reduction Rule 6.** *If a vertex* $v \in G \setminus (X \cup Y)$ *is adjacent to two vertices of the same component of* $G[Y]$*, then make* $X = X \cup \{v\}$*. The instance* $(G, X, Y, k, \ell)$ *reduces to* $(G, X \cup \{v\}, Y, k, \ell)$.

Reduction Rule 6 is safe, as $v$ has to be there in every FVS of $G \setminus X$ which is disjoint from $Y$. Otherwise, we cannot remove a cycle formed by vertices in $Y \cup \{v\}$.

Now, we start proposing the algorithm step by step. Later, we use the branching algorithm in this. We terminate either if

1. $G[Y]$ contains a cycle. Then, the instance we are dealing with is a no-instance,
2. $|X| = k$. At this stage, we check if $|N(v) \cap X| \leq \ell$ for all $v \in V(G) \setminus X$ and $G \setminus X$ is a forest. If yes, we conclude that we are dealing with a yes-instance. If no, then terminate the algorithm along that branch, or
3. graph $G \setminus X$ is a forest. If this happens, we check if $|X| \leq k$ and $|N(v) \cap X| \leq \ell$ for all $v \in V(G) \setminus X$. If yes, conclude that we are dealing with a yes-instance. If no, terminate the algorithm along that branch.

**Step 1 -** Apply Reduction Rules 4, 5 and 6 exhaustively and check the value of $|X|$ at the end. If $|X| = k$, then check 2. If $|X| > k$, conclude that we are dealing with a no-instance of Disjoint Relax-FFVS problem. Else if $|X| < k$, proceed to Step 2.

**Step 2 -** We construct a branching algorithm to solve the problem further. For a reduced instance $I = (G, X, Y, k, \ell)$, we define the measure of this branching algorithm as

$$\mu(I) = k + \gamma(Y) - |X|$$

where $\gamma(Y)$ is the total number of components in $G[Y]$ and $|X|$ is the size of set $X$. First, we branch on those vertices $v \in G \setminus (X \cup Y)$ for which $|N(v) \cap Y| \geq 2$. Observe that $v$ must be connected to different components of $G[Y]$ because of Reduction Rules 6. In the branching algorithm, either vertex $v$ goes into $X$ or it goes into $Y$. If $v$ goes into $X$, $\mu(I)$ decreases as $|X|$ increases. If $v$ goes into $Y$, then since $v$ is connected to two different components of $G[Y]$, the total number of components in $G[Y]$ now decreases, that is, value of $\gamma(Y)$ decreases and so does $\mu(I)$. At every node of the branching algorithm, an instance goes through the Reduction Rules 4, 5 and 6 and also through the checking of conditions mentioned in 1, 2 and 3. Finally, we arrive at a situation where no vertex in $V(G) \setminus (X \cup Y)$ has two or more neighbours in set $Y$. At this stage we apply the strategy given in Step 3.

**Step 3 -** Now in order to reduce the cycles in $G \setminus X$, we aim to find two nearest vertices

which have adjacent vertices in $Y$ and try to branch on the path between these two vertices. We know $F = G \setminus (X \cup Y)$ is a forest, that means, a collection of trees. Consider a rooted tree $T$ with root '$r$'. Now, select a leaf $v$ which is at the maximum distance from $r$ in $T$. Observe that $v$ is a degree-2 vertex in $G \setminus X$ and $|N(v) \cap Y| = 1$. Then find a nearest $u$ to $v$ such that $d_F(u) > 2$. Let $P = \langle v, w_1, w_2, \ldots, w_p, u \rangle$ be the path between $v$ and $u$. Now, find the nearest $w_i$ to $v$ such that $|N(w_i) \cap Y| = 1$. Here, we branch on the vertices $\{v, w_1, w_2, \ldots, w_i\}$ as either $v$ goes into $X$, $w_1$ goes into $X$, $w_i$ goes into $X$ or all $\{v, w_1, w_2, \ldots, w_i\}$ goes into $Y$. Observe that, all $\{v, w_1, w_2, \ldots, w_{i-1}\}$ are degree-2 vertices in $G \setminus X$. Hence, all of them are parts of the same collection of cycles in $G \setminus X$. Thus, it is sufficient to choose one vertex out of $\{v, w_1, w_2, \ldots, w_{i-1}\}$ into the solution set. So, without loss of generality we consider the condition that '$w_1$ goes into $X$' in the branching tree. In each case mentioned above, the measure decreases. The measure $\mu(I)$ doesn't decrease only if a cycle gets formed in $G[Y]$. In this case we terminate along this branch as mentioned in 1. So, we don't have to worry about what happens to the height of the branching tree if the measure is not decreasing.

**Step 4 -** If Step 3 is no further applicable, then we arrive at a case where path $P = \langle v, w_1, w_2, \ldots, w_p, u \rangle$ does not contain any $w_i$ such that $|N(w_i) \cap Y| = 1$. Hence all vertices $\{v, w_1, w_2, \ldots, w_p\}$ are degree two vertices in $G \setminus X$. But because of Reduction Rule 5, there can at most be three consecutive degree-2 vertices in $G \setminus X$. Thus, there are at most three vertices, namely $v$, $w_1$ and $w_2$, with degree-2 along path $P$ from $v$ to $u$. Now if $|N(u) \cap Y| = 1$, branch on the vertices $v$, $w_1$, $w_2$ and $u$ as either $v$ goes into $X$, $w_1$ goes into $X$, $u$ goes into $X$ or all $v$, $w_1$, $w_2$ and $u$ goes into $Y$. In each case the measure $\mu(I)$ decreases. It is sufficient to consider the case of $w_1$ out of all three degree-2 vertices. At each node we apply Reduction Rules 4, 5 and 6 and also check the conditions mentioned in 1, 2 and 3.

**Step 5 -** If step 4 is not applicable because $|N(u) \cap Y| = 0$, then since $d(u) > 2$, it has another branch with leaf node $v'$, but not longer than the one which contains $v$. Certainly, $|N(v') \cap Y| = 1$. Without loss of generality assume that the path $P'$ between vertices $v'$ and $u$ contains vertices $w_3$, $w_4$ such that $d(w_3) = d(w_4) = 2$ in graph $G \setminus X$. See Figure 7.1. Now, we can branch as either $w_1$ goes into $X$, $u$ goes into $X$, $w_3$ goes into $X$ or all $v$, $w_1$, $w_2$, $u$, $w_3$, $w_4$ and $v'$ goes into $Y$. Here also the measure decreases in each case.

At each node, we apply Reduction Rules 4, 5, 6 and check conditions 1, 2 and 3. We keep on doing Steps 1-5 for each tree $T$ of the forest $F$. The depth of the branching algorithm is bounded by $\mu$ which can at most be $2k$. Hence the total number of nodes in the branching

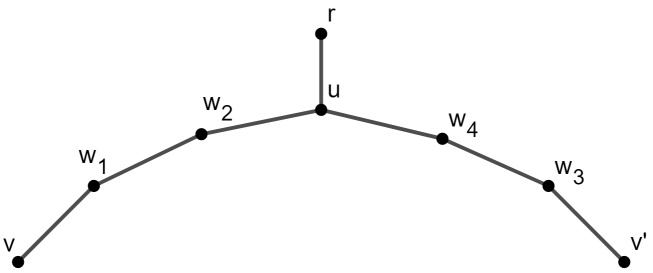



Figure 7.1: Vertices from $G \setminus X$ which are involved in the branching algorithm.

tree are bounded by $4^{2k}$, which is $16^k$. At each node, it takes polynomial time in input size to apply the Reduction Rules and check the conditions. Hence, Disjoint Relax-FFVS can be solved in $\mathcal{O}^*(16^k)$.

**Theorem 15.** *Relax-FFVS problem can be solved in time* $\mathcal{O}^*(17^k)$.

**Proof.** We begin by obtaining a feedback vertex set $S$ of $G$ of size at most $k$ in time $3.6181^k n^{\mathcal{O}(1)}$ using the technique of iterative compression [1]. This $S$ can be partitioned into $X$ and $Y$ in $\sum_{i=0}^{k} \binom{k}{i}$ ways. By lemma 14, for each partition, the disjoint Relax-FFVS problem can be solved in $\mathcal{O}^*(16^k)$.

$$\sum_{i=0}^{k} \binom{k}{i} 16^k = 17^k$$

Hence the Relax-FFVS problem can be solved in time $\mathcal{O}^*(3.6181^k + 17^k) = \mathcal{O}^*(17^k)$.

# Chapter 8

# Conclusion

This dissertation contains a study carried on computational fair deletion problems. Mainly, three variants of Fair Feedback Vertex Set problem were studied, namely, Fair Feedback Vertex Set (FFVS), Minimum Fair Feedback Vertex Set (Min-FFVS), Relaxed Fair Feedback Vertex Set (Relax-FFVS). We examined these problems in the field of Parameterized Complexity.

In Chapter 4, we showed some computational lower bounds on the complexity of the FFVS problem when parameterized by treewidth and treedepth. We proved that FFVS is $W[1]$-hard when parameterized by treedepth and treewidth. In Chapter 5, we proved it is computationally tractable with respect to neighbourhood diversity of the graph, in which we obtained an FPT algorithm running in time $3^k k^2 n^{\mathcal{O}(1)}$, where $k$ is the neighbourhood diversity of the graph.

Later, in Chapter 6, we proved that the Min-FFVS problem is NP-complete. And we obtained a kernel with respect to $(\Delta + k)$ for this problem, the kernel size is $(k + 8\Delta k - 3)$. In Chapter 7, we moved towards Relax-FFVS problem and designed an FPT algorithm running in time $\mathcal{O}^*(17^k)$ to solve the problem, considering the solution size as a parameter. It would be interesting to study if there exist a polynomial kernel of this problem with respect to the solution size.

# Bibliography


[1] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms. Springer, 2015.*

[2] Tomás Masarík, Tomás Toufar. *Parameterized complexity of fair deletion problems, arXiv:1605.07959v1 [cs.DS], 2016.*

[3] Dusan Knop, Tomás Masarík, Tomás Toufar. *Parameterized Complexity of Fair Vertex Evaluation Problems II, arXiv:1803.06878v1 [cs.CC], 2018.*

[4] Richard M. Karp, *Reducibility Among Combinatorial Problems, Complexity of Computer Computations, 1972, Pages 85-103.*

[5] L. Lin and S. Sahni, *Fair edge deletion problems, IEEE Transactions on Computers, Volume 38 Issue 5, 1989, Pages 756-761.*

[6] Luca Tevison, *Lecture notes on computational complexity, Notes written in Fall* (2002) https://people.eecs.berkeley.edu/ luca/notes/complexitynotes02.pdf

[7] Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, Meirav Zehavi, *Kernelization: Theory of Parameterized Preprocessing, 10.1017/9781107415157, 2018.*

[8] Jon Kleinberg, Eva Tardos *Algorithm Design, Addison-Wesley, 2005.*

[9] Douglas B. West, *Introduction to Graph Theory, Prentice Hall, 2000.*

[10] Robert Ganian, *Using Neighborhood Diversity to Solve Hard Problems, arXiv:1201.3091v2 [cs.DS], 2012.*

[11] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein *Introduction to Algorithms, Third Edition, MIT press, 2009.*