

Alliances in Graphs: A Parameterized Perspective

A Thesis

submitted to

Indian Institute of Science Education and Research Pune
in partial fulfillment of the requirements of the degree of

Doctor of Philosophy

by

Shuvam Kant Tripathi



Indian Institute of Science Education and Research Pune

Dr. Homi Bhabha Road,
Pashan, Pune 411008, INDIA.

July, 2022

Supervisor: Dr. Soumen Maity, IISER Pune

© Shuvam Kant Tripathi 2022

All rights reserved

Certificate

Certified that the work incorporated in the thesis entitled Alliances in Graphs: A Parameterized Perspective, submitted by Shuvam Kant Tripathi was carried out by the candidate, under my supervision. The work presented here or any part of it has not been included in any other thesis submitted previously for the award of any degree or diploma from any other university or institution.



Dr. Soumen Maity, IISER Pune

Committee:

Dr. Soumen Maity, IISER Pune

Dr. Vivek Mallick, IISER Pune

Professor Saket Saurabh, IMSc Chennai

This thesis is dedicated to my parents.

Declaration

I declare that this written submission represents my ideas in my own words and where others' ideas have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that violation of the above will be cause for disciplinary action by the institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.



Shuvam Kant Tripathi

Acknowledgments

It is a great pleasure to express my deep sense of gratitude to Dr. Soumen Maity for supervising my research work and for the constant encouragement I received from him.

I am thankful to Professor Saket Saurabh and Dr. Vivek Mallick for being in my research advisory committee and for their valuable suggestions. I am grateful to Ajinkya Gaikwad for continuous discussion, encouragement and joint papers in this thesis. Thanks are also due to our research group members Hitendra Kumar and Mihir Neve.

I am grateful to the Indian Institute of Science Education and Research (IISER) Pune for providing ample facilities for research. Finally I wish to express my thanks to my parents and my bother for their encouragement during the course of my work.



Shuvam Kant Tripathi

Abstract

Throughout history, humans have formed communities, guilds, faiths etc in the hope of coming together with a group of people having similar requirements, visions and goals. Their reasons to do so, usually rest on the fact that any group with common interests often provides added mutual benefits to the union in fields of trade, culture, defense, etc as compared to the individual. Such activities are commonly seen in the present day, in areas of geo-politics, cultures, trades, economics, unions etc and are popularly termed as *alliances*. The concept of an alliance was first captured as a graph theoretic problem in 2000 by Kristiansen, Hedetniemi, and Hedetniemi in [45]. Based on the structure, formation and goals of an alliance, many variations of the problem exist in graph theory. A defensive alliance is usually formed with the aim of defending its members against non-members, and hence it is natural to ask that each member of the alliance should have more friends within the alliance (including oneself) than outside. More formally, a *defensive alliance* in a graph $G = (V, E)$ is a non-empty set of vertices S satisfying the condition that every vertex $v \in S$ has at least as many neighbors (including itself) in S than it has in $V \setminus S$. Similarly, an offensive alliance is formed with the inverse goal of offending or attacking non-members of the alliance. An *offensive alliance* in a graph $G = (V, E)$ is a non-empty set of vertices S satisfying the condition that every vertex $v \in N(S)$ has at least as many neighbors in S than it has in $V \setminus S$ (including itself). It is known that the problems of finding small defensive and offensive alliances are NP-complete. We enhance our understanding of the problems from the viewpoint of parameterized complexity. *Strong* versions of the above problems do not consider the self to be a friend, and the *minimal* versions try to find those alliances which lose the required property in the absence of any member.

These variations in types of alliances manifest themselves ubiquitously in daily life scenarios and applications, thereby attributing value to the study of their graph theoretic versions and related algorithms. Alliances have been used to study problems such as classification and clustering problems, understanding communities on the internet, protocols for distribution etc. [19, 56, 34]. The data clustering problem relies on the concept of partitioning the vertices of the graph into multiple strong defensive alliances. This problem was introduced by Gerber and Kobler [32] and is called the SATISFACTORY PARTITION problem.

Almost all variants of the alliance problem are NP-hard, and so there is very little hope for finding efficient polynomial time algorithms for these problems. However, the applicability

of this problem warrants the use of techniques which can make these variants computationally tractable under certain scenarios. Two key approaches for achieving this goal involve the polynomial-time approximation algorithms and the concept of fixed-parameter tractable (FPT) algorithms. The former involves finding algorithms which approximate the optimal solution in polynomial time, thereby creating a trade-off between the correctness and time complexity of the algorithm. On the other hand, FPT algorithms originated from the theory of parameterized complexity as introduced by Downey and Fellows. Problems which do not admit a polynomial time solution, often make use of exponential time algorithms, thereby making them intractable. The idea in these cases is to associate a parameter k with every instance of the problem, and then try to develop an algorithm that captures the exponential growth only as a function of the parameter k . We would then obtain an algorithm with running time $f(k) \cdot n^c$ where n is the size of the input, k is the parameter, $f(\cdot)$ is some computable function, and c is a constant that is independent on k and n . Such algorithms are said to be *fixed-parameter tractable* algorithms. However, presently, FPT algorithms are not known for all NP-hard problems. The study of these fixed-parameter intractable problems led to the formation of the W-hierarchy of complexity classes. Similar to the theory of classical complexity, showing that a parameterised problem is one of the *hardest* problems in any one of the W-classes, gives evidence that it is unlikely for the problem to admit an FPT algorithm.

In this dissertation, we consider parameterized algorithms and complexity of the following problems: defensive and offensive alliances in graphs, locally minimal defensive alliances in graphs, and the satisfactory partition problem in graphs. We obtain the following results.

- We design FPT algorithms for DEFENSIVE ALLIANCE and OFFENSIVE ALLIANCE when parameterized by neighbourhood diversity of the input graph. We prove that DEFENSIVE ALLIANCE and OFFENSIVE ALLIANCE are polynomial time solvable for graphs with bounded treewidth.
- We study parameterized intractability of the DEFENSIVE ALLIANCE problem. We prove that the DEFENSIVE ALLIANCE problem is W[1]-hard when parameterized by the pathwidth of the input graph. We also prove that the EXACT DEFENSIVE ALLIANCE problem is W[1]-hard when parameterized by the feedback vertex set number and the pathwidth of the input graph.
- We design a polynomial-time algorithm for the CONNECTED LOCALLY MINIMAL

STRONG DEFENSIVE ALLIANCE on trees. We prove that LOCALLY MINIMAL DEFENSIVE ALLIANCE problem is NP-complete, even when restricted to planar graphs. We give a randomized FPT algorithm for the EXACT CONNECTED LOCALLY MINIMAL DEFENSIVE ALLIANCE problem using color coding technique. We give an FPT algorithm for LOCALLY MINIMAL DEFENSIVE ALLIANCE when parameterized by neighbourhood diversity of the input graph. We prove that EXACT CONNECTED LOCALLY MINIMAL DEFENSIVE ALLIANCE parameterized by treewidth is $W[1]$ -hard and thus not FPT (unless $FPT=W[1]$). Finally we show that the LOCALLY MINIMAL DEFENSIVE ALLIANCE problem is polynomial time solvable for graphs with bounded treewidth.

- We design a polynomial-time algorithm for the SATISFACTORY PARTITION problem for block graphs. We prove that the SATISFACTORY PARTITION and BALANCED SATISFACTORY PARTITION problems are fixed parameter tractable (FPT) when parameterized by neighbourhood diversity. We show that the SATISFACTORY PARTITION and BALANCED SATISFACTORY PARTITION problems can be solved in polynomial time for graphs of bounded clique-width. Finally we prove that the BALANCED SATISFACTORY PARTITION problem is $W[1]$ -hard when parameterized by treewidth.

Contents

Abstract	xi
1 Introduction	1
1.1 Defensive and Offensive Alliances in Graphs	1
1.2 Locally Minimal Defensive Alliance	5
1.3 The Satisfactory Partition Problem	8
1.4 Parameterized Complexity	11
1.5 Literature Survey	12
1.6 Graph Theory	15
1.7 Structural Graph Parameters	17
1.8 An Overview of the Thesis	24
2 Introduction to Parameterized Complexity	27
2.1 Fixed Parameter Tractability	27
2.2 Fixed-Parameter Intractability	38
3 Defensive and Offensive Alliances	45
3.1 Introduction	45
3.2 FPT algorithm parameterized by neighbourhood diversity	45

3.3	FPT algorithm parameterized by domino treewidth	56
3.4	Graphs of bounded treewidth	60
4	Parameterized Intractability of Defensive Alliance Problem	67
4.1	Introduction	67
4.2	W[1]-hardness parameterized by pathwidth	68
5	Locally Minimal Defensive Alliance	85
5.1	Introduction	85
5.2	Polynomial-Time Algorithm for Connected Locally Minimal Strong Defensive Alliance on Trees	86
5.3	Locally Minimal Defensive Alliance in Planar Graphs is NP-complete	92
5.4	A color coding algorithm for EXACT CONNECTED LOCALLY MINIMAL DEFENSIVE ALLIANCE	95
5.5	FPT algorithm parameterized by neighbourhood diversity	96
5.6	Hardness of EXACT CONNECTED LOCALLY MINIMAL DEFENSIVE ALLIANCE parameterized by treewidth	100
5.7	Graphs of bounded treewidth	111
6	The Satisfactory Partition Problem	117
6.1	Introduction	117
6.2	Polynomial Time Algorithm for Block Graphs	118
6.3	FPT algorithm parameterized by neighbourhood diversity	122
6.4	Graphs of bounded clique-width	125
6.5	Hardness of BALANCED SATISFACTORY PARTITION parameterized by treewidth	128
7	Conclusions and Open Problems	141

Chapter 1

Introduction

In this dissertation, we consider parameterized algorithms and complexity of the following graph problems: defensive and offensive alliances in graphs, locally minimal defensive alliances in graphs, and the satisfactory partition problem in graphs. We give below, section-wise, the problems considered.

1.1 Defensive and Offensive Alliances in Graphs

Throughout history, humans have formed communities, guilds, faiths etc in the hope of coming together with a group of people having similar requirements, visions and goals. Their reasons to do so, usually rest on the fact that any group with common interests often provides added mutual benefits to the union in fields of trade, culture, defense, etc as compared to the individual. Such activities are commonly seen in the present day, in areas of geo-politics, cultures, trades, economics, unions etc and are popularly termed as *alliances*. The idea of an alliance has been applied to study several models like distributed protocols, classification or clustering problems, and studying communities on the internet [34, 19, 56].

A defensive alliance is an alliance formed with the main purpose of defending its members, while an offensive alliance gets formed with the purpose of attacking non-members. The former desires having more friends within the alliance, while the latter happens when a non-member has more enemies in the alliance than friends outside. These notions motivate

the study of defensive and offensive alliances in graphs. A defensive alliance which is also a dominating set is called a *global alliance*. Kristiansen, Hedetniemi, and Hedetniemi [45] pioneered the study of various types of alliance in graphs including defensive, offensive and powerful alliances. Since then, many variations of the alliance problem, [21, 53, 10, 51, 55], including generalisations called r -alliances [52], have been extensively researched upon.

Throughout this dissertation, $G = (V, E)$ denotes a finite, simple and undirected graph of order $|V| = n$. The *subgraph induced* by $S \subseteq V(G)$ is denoted by $G[S]$. For a vertex $v \in V$, we use $N(v) = \{u : (u, v) \in E(G)\}$ to denote the (*open*) *neighbourhood* of vertex v in G , and $N[v] = N(v) \cup \{v\}$ to denote the *closed neighbourhood* of v . The degree $d(v)$ of a vertex $v \in V(G)$ is $|N(v)|$. For a subset $S \subseteq V(G)$, we define its closed neighbourhood as $N[S] = \bigcup_{v \in S} N[v]$ and its open neighbourhood as $N(S) = N[S] \setminus S$. For a non-empty subset $S \subseteq V$ and a vertex $v \in V(G)$, $N_S(v)$ denotes the set of neighbours of v in S , that is, $N_S(v) = \{u \in S : (u, v) \in E(G)\}$. We use $d_S(v) = |N_S(v)|$ to denote the number of neighbours of v in S . The *complement* of the vertex set S in V is denoted by S^c . Now we give definitions of defensive alliance and defensive r -alliance.

Definition 1.1.1. A non-empty set $S \subseteq V$ is a *defensive alliance* in $G = (V, E)$ if $d_S(v) + 1 \geq d_{S^c}(v)$ for all $v \in S$.

Definition 1.1.2. For an integer r , a non-empty set $S \subseteq V$ is a *defensive r -alliance* in G if for each $v \in S$, $d_S(v) \geq d_{S^c}(v) + r$. A set is a defensive alliance if it is a defensive (-1) -alliance.

We often use the terms *defenders* and *attackers* of an element v of a defensive alliance S . By these we mean the sets $N[v] \cap S$ and $N[v] \setminus S$, respectively. A vertex $v \in S$ is said to be *protected* if the number of defenders of v is greater than or equal to the number of attackers of v , that is, $|N[v] \cap S| = d_S(v) + 1 \geq d_{S^c}(v) = |N[v] \setminus S|$. A set $S \subseteq V$ is a defensive alliance if every vertex in S is protected. We define DEFENSIVE ALLIANCE and EXACT DEFENSIVE ALLIANCE as follows:

DEFENSIVE ALLIANCE

Input: An undirected graph $G = (V, E)$ and an integer $k \geq 1$.

Question: Is there a defensive alliance $S \subseteq V(G)$ such that $|S| \leq k$?

While the DEFENSIVE ALLIANCE problem asks for defensive alliance of size at most k , we also consider the EXACT DEFENSIVE ALLIANCE problem that concerns defensive alliance of size exactly k .

EXACT DEFENSIVE ALLIANCE

Input: An undirected graph $G = (V, E)$ and an integer $k \geq 1$.

Question: Is there a defensive alliance $S \subseteq V(G)$ such that $|S| = k$?

Now we give definitions of offensive alliance and offensive r -alliance.

Definition 1.1.3. A non-empty set $S \subseteq V$ is an *offensive alliance* in G if $d_S(v) \geq d_{S^c}(v) + 1$ for all $v \in N(S)$.

Definition 1.1.4. For an integer r , a non-empty set $S \subseteq V$ is an *offensive r -alliance* in G if for each $v \in N(S)$, $d_S(v) \geq d_{S^c}(v) + r$. An offensive 1-alliance is called an offensive alliance.

Informally, given a graph $G = (V, E)$, we say a set S is an offensive alliance if every vertex that is adjacent to S is outgunned by S ; more of its neighbours are in S than outside S . Equivalently, since an attack by the vertices in S on the vertices in $V \setminus S$ can result in no worse than a “tie” for S , we say that S can effectively attack $N(S)$. For example, Figure 1.1(i) shows a minimum size defensive alliance in G and Figure 1.1(ii) shows a minimum size offensive alliance in G .

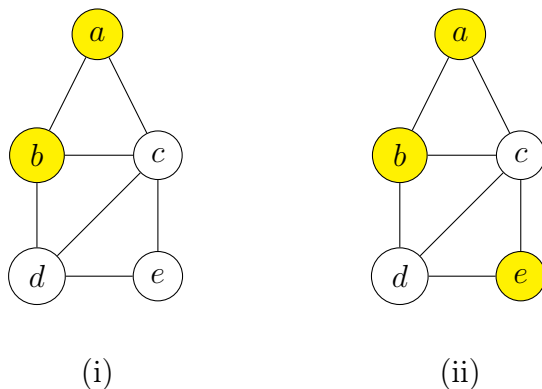


Figure 1.1: (i) $S = \{a, b\}$ is a minimum size defensive alliance in G , (ii) $S = \{a, b, e\}$ is a minimum size offensive alliance in G .

Definition 1.1.5. A non-empty set $S \subseteq V$ is a *strong offensive alliance* in G if $d_S(v) \geq d_{S^c}(v) + 2$ for all $v \in N(S)$.

In this thesis, we consider OFFENSIVE ALLIANCE and STRONG OFFENSIVE ALLIANCE under structural parameters. We define these problems as follows:

OFFENSIVE ALLIANCE
Input: An undirected graph $G = (V, E)$ and an integer $k \geq 1$.
Question: Is there an offensive alliance $S \subseteq V(G)$ such that $|S| \leq k$?

STRONG OFFENSIVE ALLIANCE
Input: An undirected graph $G = (V, E)$ and an integer $k \geq 1$.
Question: Is there a strong offensive alliance $S \subseteq V(G)$ such that $|S| \leq k$?

It was shown that the problem of finding a minimum alliance of any variant is NP-hard in general. It is known that the DEFENSIVE ALLIANCE problem is NP-complete [38]. The defensive r -alliance [52] and global defensive r -alliance problems [17] are NP-complete for any fixed r . The defensive alliance problem is NP-complete even when restricted to split, chordal and bipartite graph [39]. Fernau et al. showed that the offensive r -alliance and global offensive r -alliance problems are NP-complete for any fixed r [18]. There are polynomial time algorithms for finding minimum alliances in trees [9, 38, 39]. A polynomial time algorithm for finding minimum defensive alliance in series parallel graph is presented in [38]. There has also been some work on the parameterized complexity of alliance problems. It is known that DEFENSIVE ALLIANCE and OFFENSIVE ALLIANCE are fixed-parameter tractable when parameterized by the solution size [13, 16]. Alliance problems have been studied with respect to structural graph parameters. The alliance problems for all variants are efficiently solvable for much larger graph classes. Kiyomia and Otachi [42] proved that alliance problems can be solved in polynomial time for graphs of bounded clique-width. They also showed that the problems are fixed-parameter tractable when parameterized by the vertex cover number. Ensico [13] showed that the problems of finding minimum size defensive alliances and global defensive alliances are fixed-parameter tractable when parameterized by both treewidth and maximum degree. Bliem and Woltran [6] proved that defensive alliance problem is W[1]-hard when parameterized by the treewidth of the input graph. It is also known that the OFFENSIVE ALLIANCE problem is W[1]-hard when parameterized by the treewidth of the input graph [22]. This puts these two problems among the few problems that are FPT when

parameterized by solution size but not when parameterized by treewidth (unless $\text{FPT}=\text{W}[1]$). We design FPT algorithms for DEFENSIVE ALLIANCE and OFFENSIVE ALLIANCE when parameterized by neighbourhood diversity of the input graph. We prove that OFFENSIVE ALLIANCE is FPT when parameterized by domino treewidth of the input graph. We also prove that DEFENSIVE ALLIANCE and OFFENSIVE ALLIANCE are polynomial time solvable for graphs with bounded treewidth.

1.2 Locally Minimal Defensive Alliance

In the previous section we defined the problem of DEFENSIVE ALLIANCE. This problem has been extensively studied, both combinatorially and computationally, over the course of the last 20 years. Recall that an alliance was said to be defensive, in a graph, if every member has at least as many friends within the alliance (including itself) as compared to those not in the alliance. If oneself is not treated as a friend, then the resulting defensive alliance is said to be a *strong defensive alliance*.

Forming an alliance usually requires a lot of effort for it to be profitably beneficial. In the same sense, it is a crucial problem to determine the *minimum* number of people required to form a defensive alliance. On the other hand, larger alliances are often more powerful, but are harder to form. This sort of trade-off between size and ease of formation of a defensive alliance can be mitigated by studying the concepts of *maximum-sized minimal alliances*, that is, those alliances of the maximum size for which, each member is essential in the sense that the alliance does not stay defensive in his absence. Observe that this concept is based on the fact that DEFENSIVE ALLIANCE is not a hereditary graph problem. Indeed, removing a vertex from the defensive alliance can cause a member to lose a friend within the alliance, thereby leaving him undefended. Hence a subset of a defensive alliance need not be defensive.

The above ideas motivate the need to study problems like the *locally minimal defensive alliance* and the *globally minimal defensive alliance*. The former, as referred to by Shafique, [34] is a defensive alliance which does not stay defensive if any single member gets removed. On the other hand, the latter, as studied by Bazgan et al. [1] considers those defensive alliances, for which no proper subset stays defensive. Motivated by the trade-off given above, we shall study the problem of finding *locally minimal defensive alliances of maximum size*. There is also a general mathematical interest in such type of problems [48].

Throughout this thesis, $G = (V, E)$ denotes a finite, simple and undirected graph of order $|V| = n$. A non-empty set $S \subseteq V$ is a *defensive alliance* in G if for each $v \in S$, $|N[v] \cap S| \geq |N(v) \setminus S|$, or equivalently, $d_S(v) + 1 \geq d_{S^c}(v)$. A vertex $v \in S$ is said to be *protected* if $d_S(v) + 1 \geq d_{S^c}(v)$. Here v has $d_S(v) + 1$ defenders and $d_{S^c}(v)$ attackers in G . A set $S \subseteq V$ is a defensive alliance if every vertex in S is protected.

Definition 1.2.1. A vertex $v \in S$ is said to be *marginally protected* if it becomes unprotected when any of its neighbour in S is moved from S to $V \setminus S$. A vertex $v \in S$ is said to be *overprotected* if it remains protected even when any of its neighbours is moved from S to $V \setminus S$.

Definition 1.2.2. [1] An alliance S is called a *locally minimal alliance* if for any $v \in S$, $S \setminus \{v\}$ is not an alliance.

It is important to note that if S is a locally minimal defensive alliance, then for every vertex $v \in S$, at least one of its neighbours in S is marginally protected.

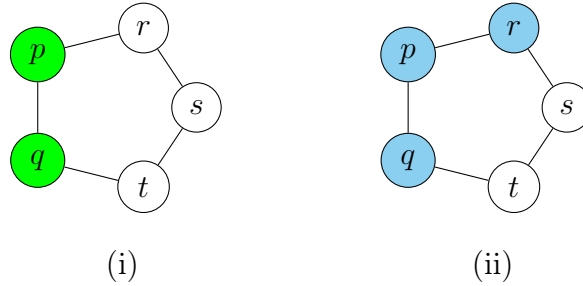


Figure 1.2: (i) $S = \{p, q\}$ forms a locally minimal defensive alliance of size 2, (ii) $S = \{p, q, r\}$ is not a locally minimal defensive alliance.

In Figure 1.2(i), $S = \{p, q\}$ forms a locally minimal defensive alliance. Here both p and q are marginally protected. Thus p has a marginally protected neighbours q , similarly q has a marginally protected neighbours p . In Figure 1.2(ii), $S = \{p, q, r\}$ is a defensive alliance but not a locally minimal defensive alliance. Here p and r are marginally protected but q is overprotected. Thus p and r do not have a marginally protected neighbours; and hence S is not a locally minimal defensive alliance. In other words, $S = \{p, q, r\}$ is not a locally minimal defensive alliance as $S \setminus \{r\} = \{p, q\}$ is a defensive alliance.

Definition 1.2.3. [1] An alliance S is *globally minimal alliance* or shorter *minimal alliance* if no proper subset is an alliance.

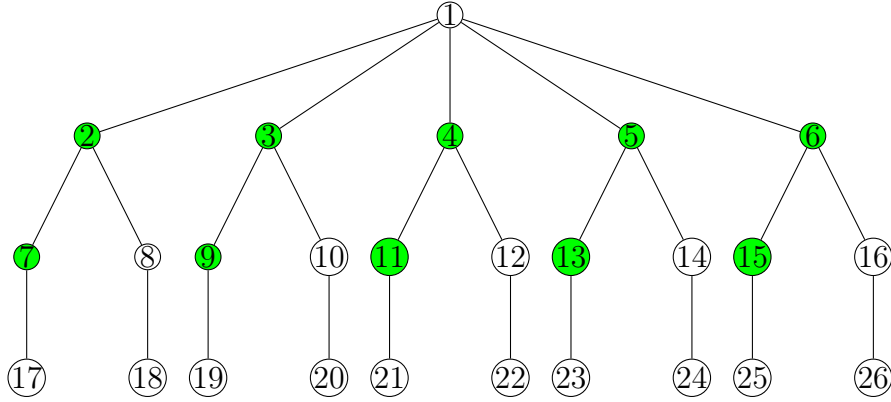


Figure 1.3: The set $S_1 = \{7, 2, 9, 3, 11, 4, 13, 5, 15, 6\}$ is a locally minimal defensive alliance of size 10 and $S_2 = \{1, 2, 3\}$ is a globally minimal defensive alliance of size 3 in G .

A defensive alliance S is connected if the subgraph induced by S is connected. An alliance S is called a *connected locally minimal alliance* if for any $v \in S$, $S \setminus \{v\}$ is not a connected alliance. Notice that any globally minimal alliance is also connected. We consider **LOCALLY MINIMAL DEFENSIVE ALLIANCE** and **EXACT CONNECTED LOCALLY MINIMAL DEFENSIVE ALLIANCE** under structural parameters. We define the problems as follows:

LOCALLY MINIMAL DEFENSIVE ALLIANCE
Input: An undirected graph $G = (V, E)$ and an integer $k \leq |V(G)|$.
Question: Is there a locally minimal defensive alliance $S \subseteq V(G)$ such that $|S| \geq k$?

EXACT CONNECTED LOCALLY MINIMAL DEFENSIVE ALLIANCE
Input: An undirected graph $G = (V, E)$ and an integer $k \leq |V(G)|$.
Question: Is there a connected locally minimal defensive alliance $S \subseteq V(G)$ such that $|S| = k$?

Bazgan, Fernau and Tuza [1] proved that deciding if a graph contains a locally minimal strong defensive alliance of size at least k is NP-complete, even when restricted to bipartite graphs with average degree less than 3.6; deciding if a graph contains a locally minimal defensive alliance of size at least k , even for bipartite graphs with average degree less than 5.6. The authors also proved that deciding if a graph contains a connected locally minimal strong defensive alliance or a connected locally minimal defensive alliance of size at least

k is NP-complete, even when restricted to bipartite graphs with average degree less than $2 + \epsilon$, for any $\epsilon > 0$. We design a polynomial-time algorithm for the CONNECTED LOCALLY MINIMAL STRONG DEFENSIVE ALLIANCE on trees. We prove that LOCALLY MINIMAL DEFENSIVE ALLIANCE problem is NP-complete, even when restricted to planar graphs. We give a randomized FPT algorithm for the EXACT CONNECTED LOCALLY MINIMAL DEFENSIVE ALLIANCE problem using color coding technique. We give an FPT algorithm for LOCALLY MINIMAL DEFENSIVE ALLIANCE when parameterized by neighbourhood diversity of the input graph. We prove that EXACT CONNECTED LOCALLY MINIMAL DEFENSIVE ALLIANCE parameterized by treewidth is W[1]-hard and thus not FPT (unless FPT=W[1]). Finally we show that the LOCALLY MINIMAL DEFENSIVE ALLIANCE problem is polynomial time solvable for graphs with bounded treewidth.

1.3 The Satisfactory Partition Problem

While the formation of an alliance is highly beneficial to its members, it can have adverse effects on the non-members of the alliance. Such an alliance can create a sense of disharmony within the community, and can be harmful in the long run. Thus, while the formation of groups and communities itself cannot be prevented, one can hope that the groups created are such that they partition the entire population into beneficial defensive alliances. With this in mind, we shall study a graph theoretic version of this scenario, where we try to partition the vertices of a graph into defensive or strong defensive alliances, and try to characterise the set of graphs for which such a partition exists.

This problem also has applications in similarity networks of natural objects [20]. Given a set of objects, we can create a similarity graph or network, which links two objects based on their similarities. Then, the problem would be to partition these objects into smaller classes or clusters such that every object has more similarity with members of the same class, than other classes. Finding such clusters in a given data set automatically by a computer is a popular line of study in the present day. Cluster recognition finds applications in many fields including those of pattern recognition, information theory and behavioural sciences [34].

Similar to the partitioning problem above, Gerber and Kobler [32] defined a *satisfactory partition* to be a partition of the vertices of a graph into two non-empty parts, such that each part forms a strong defensive alliance. A graph for which a satisfactory partition exists

is said to be *satisfactorily partitionable*. Complete graphs, star graphs, complete bipartite graphs with at least one part having odd size are some examples of graphs which are not satisfactorily partitionable, while non-star trees and cycles of length at least 4 are some graphs for which a satisfactory partition can be easily found [5]. See Figure 1.4.

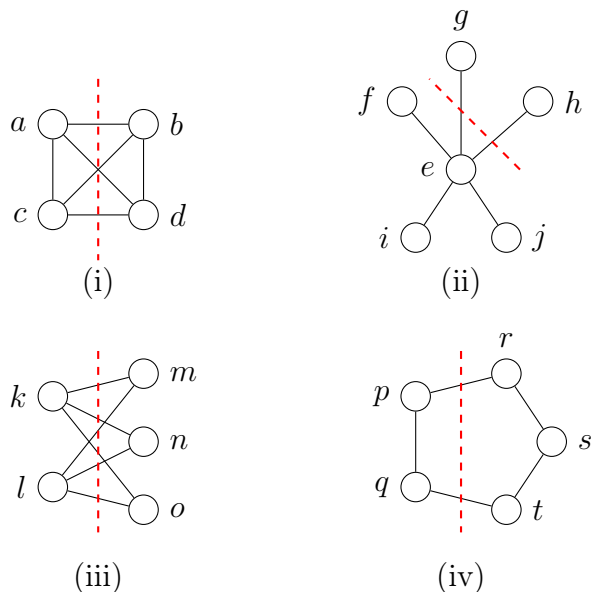


Figure 1.4: (i) An unsatisfactory partition of K_4 (ii) An unsatisfactory partition of star graph (iii) An unsatisfactory partition of $K_{2,3}$ (iv) A satisfactory partition of C_5 .

We study the parameterized complexity of SATISFACTORY PARTITION and BALANCED SATISFACTORY PARTITION problems. We define these problems as follows:

SATISFACTORY PARTITION
Input: A graph $G = (V, E)$.
Question: Is there a nontrivial partition (V_1, V_2) of V such that for every $v \in V$, if $v \in V_i$ then $d_{V_i}(v) \geq d_{V_{3-i}}(v)$?

A variant of this problem where the two parts have equal size is:

BALANCED SATISFACTORY PARTITION
Input: A graph $G = (V, E)$ on an even number of vertices.
Question: Is there a nontrivial partition (V_1, V_2) of V such that $|V_1| = |V_2|$ and for every $v \in V$, if $v \in V_i$ then $d_{V_i}(v) \geq d_{V_{3-i}}(v)$?

Given a partition (V_1, V_2) , we say that a vertex $v \in V_i$ is *satisfied* if $d_{V_i}(v) \geq d_{V_{3-i}}(v)$, or equivalently if $d_{V_i}(v) \geq \lceil \frac{d(v)}{2} \rceil$. A graph admitting a non-trivial partition where all vertices are satisfied is called *satisfactory partitionable*, and such a partition is called *satisfactory partition*. In Figure 1.4(i), the partition (V_1, V_2) where $V_1 = \{a, c\}$ and $V_2 = \{b, d\}$ is not a satisfactory partition. Here a has only one neighbour c in V_1 and two neighbours b, d in V_2 . Similarly the partitions in Figure 1.4(ii) and Figure 1.4(iii) are not satisfactory. However, the partition in Figure 1.4(iv) is satisfactory. Additionally, if $|V_1| = |V_2|$ holds, then it will be called a *balanced satisfactory partition* and the graph G is *balanced satisfactory partitionable*. It is easy to see that cycles of even length and complete bipartite graphs with both vertex classes of even size are trivially balanced partitionable [5].

In the first paper on this topic, Gerber and Kobler [32] considered a generalized version of this problem by introducing weights for the vertices and edges and showed that a general version of the problem is strongly NP-complete. For the unweighted version, they presented some sufficient conditions for the existence of a solution. This problem was further studied in [2, 33, 31]. The SATISFACTORY PARTITION problem is NP-complete and this implies that BALANCED SATISFACTORY PARTITION problem is also NP-complete via a simple reduction in which we add new dummy vertices and dummy edges to the graph [3, 5]. Both problems are solvable in polynomial time for graphs with maximum degree at most 4 [5]. They also studied generalizations and variants of this problem when a partition into $k \geq 3$ nonempty parts is required. Bazgan, Tuza, and Vanderpooten [2, 4] studied an “unweighted” generalization of SATISFACTORY PARTITION, where each vertex v is required to have at least $s(v)$ neighbours in its own part, for a given function s representing the degree of satisfiability. Obviously, when $s = \lceil \frac{d}{2} \rceil$, where d is the degree function, they obtained satisfactory partition. They gave a polynomial-time algorithm for graphs of bounded treewidth which decides if a graph admits a satisfactory partition, and gives such a partition if it exists. Except this result, to the best of our knowledge, the parameterized complexity of the SATISFACTORY PARTITION and BALANCED SATISFACTORY PARTITION problems have not been studied before. We design a polynomial-time algorithm for the SATISFACTORY PARTITION problem for block graphs. We prove that the SATISFACTORY PARTITION and BALANCED SATISFACTORY PARTITION problems are fixed parameter tractable (FPT) when parameterized by neighbourhood diversity. We show that the SATISFACTORY PARTITION and BALANCED SATISFACTORY PARTITION problems can be solved in polynomial time for graphs of bounded clique-width. Finally we prove that the BALANCED SATISFACTORY PARTITION problem is W[1]-hard when parameterized by treewidth.

1.4 Parameterized Complexity

A *parameterized problem* is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a fixed, finite alphabet. For an instance $(x, k) \in \Sigma^* \times \mathbb{N}$, k is called the *parameter*. For example, an instance of VERTEX COVER parameterized by the solution size is a pair (G, k) , where we expect G to be an undirected graph encoded as a string over Σ , and k is a positive integer. A pair (G, k) belongs to the VERTEX COVER parameterized language if and only if the string G correctly encodes an undirected graph (G) and moreover the graph G contains a vertex cover on k vertices.

Definition 1.4.1. A parameterized problem $L \subseteq \Sigma^* \times \mathbb{N}$ is *fixed-parameter tractable* (FPT in short) if there is an algorithm (called an FPT algorithm) that correctly decides, for input $(x, k) \in \Sigma^* \times \mathbb{N}$, whether $(x, k) \in L$ in time $f(k)n^c$ where f is some (usually computable) function, n is the size of the main part of the input x , that is $|x| = n$ and c is a constant independent of k . A complexity class containing all fixed-parameter tractable problems is called FPT.

Fixed-parameter tractable algorithms (FPT-algorithms) are helpful in solving real world problems that are in general NP-hard, but where most instances of interest have small parameter values. This is the case for many practical problems such as multiple sequence alignment in computational biochemistry, known to be equivalent to the vertex cover problem, which has an FPT-algorithm with running time $O(kn + 1.2738^k)$.

We now define the complexity class XP.

Definition 1.4.2. A parameterized problem $L \subseteq \Sigma^* \times \mathbb{N}$ is *slice-wise polynomial* (XP in short) if there is an algorithm (called an XP algorithm) that correctly decides, for input $(x, k) \in \Sigma^* \times \mathbb{N}$, whether $(x, k) \in L$ in time $f(k)n^{g(k)}$ where f and g are some (usually computable) functions, n is the size of the main part of the input x , that is $|x| = n$ and c is a constant independent of k . A complexity class containing all slice-wise polynomial problems is called XP.

Parameterized complexity classes are defined with respect to *fpt-reducibility*. A parameterized problem \mathcal{P} is *fpt-reducible* to \mathcal{Q} if in time $f(k) \cdot |(x, k)|^c$, one can transform an instance (x, k) of \mathcal{P} into an instance (x', k') of \mathcal{Q} such that $(x, k) \in \mathcal{P}$ if and only if $(x', k') \in \mathcal{Q}$, and $k' \leq g(k)$, where f and g are computable functions depending only on k . Owing to the def-

inition, if \mathcal{P} *fpt-reduces* to \mathcal{Q} and \mathcal{Q} is fixed-parameter tractable then \mathcal{P} is fixed-parameter tractable as well.

What makes the theory more interesting is a hierarchy of intractable parameterized problem classes above FPT which helps in distinguishing those problems that are not fixed parameter tractable. Central to parameterized complexity is the following hierarchy of complexity classes, defined by the closure of canonical problems under *fpt-reductions*: $\text{FPT} \subseteq \text{W}[1] \subseteq \text{W}[2] \subseteq \dots \subseteq \text{XP}$. All inclusions are believed to be strict. In particular, $\text{FPT} \neq \text{W}[1]$ under the Exponential Time Hypothesis [37]. The class $\text{W}[1]$ is the analog of NP in parameterized complexity. A major goal in parameterized complexity is to distinguish between parameterized problems which are in FPT and those which are *W[1]-hard*, i.e., those to which every problem in $\text{W}[1]$ is *fpt-reducible*. There are many problems shown to be complete for $\text{W}[1]$, or equivalently *W[1]-complete*, including the MULTICOLORED CLIQUE (MCC) problem [12].

Closely related to fixed-parameter tractability is the notion of preprocessing. A *reduction to a problem kernel*, or equivalently, *problem kernelization* means to apply a data reduction process in polynomial time to an instance (x, k) such that for the reduced instance (x', k') it holds that (x', k') is equivalent to (x, k) , $|x'| \leq g(k)$ and $k' \leq g(k)$ for some computable function g only depending on k . Such a reduced instance is called a *problem kernel*. It is easy to show that a parameterized problem is in FPT if and only if there is kernelization algorithm. A *polynomial kernel* is a kernel, whose size can be bounded by a polynomial in the parameter. We refer to [11, 12] for further details on parameterized complexity.

1.5 Literature Survey

In real life, an alliance is a collection of people, groups, or states such that the union is stronger than individual. The alliance can be either to achieve some common purpose, to protect against attack, or to assert collective will against others. This motivates the definitions of defensive and offensive alliances in graphs. The properties of alliances in graphs were first studied by Kristiansen, Hedetniemi, and Hedetniemi [45]. They introduced the concepts of defensive and offensive alliances, global offensive and global defensive alliances and studied alliance numbers of some classes of graphs such as cycles, wheels, grids and complete graphs. A set $X \subset V(G)$ is called a *dominating set* if every vertex in $V(G) \setminus X$ has at least one neighbour in X . An alliance is *global* if it is a dominating set. The alliance problems

have been studied extensively during last fifteen years [21, 53, 10, 51, 55], and generalizations called r -alliances are also studied [52]. The global defensive (offensive) alliance number of G is the cardinality of a minimum size global defensive (offensive) alliance in G . Haynes, Hedetniemi and Henning [35] studied the global defensive alliance numbers of different classes of graphs. They gave lower bounds for general graphs, bipartite graphs and trees, and upper bounds for general graphs and trees. Rodriguez-Velazquez and Sigarreta [54] studied the defensive alliance number and the global defensive alliance number of line graphs. The decision version for several types of alliances have been shown to be NP-complete. It is known that the DEFENSIVE ALLIANCE problem is NP-complete [38]. The defensive r -alliance [52] and global defensive r -alliance problems [17] are NP-complete for any fixed r . The defensive alliance problem is NP-complete even when restricted to split, chordal and bipartite graph [39]. Fernau et al. showed that the offensive r -alliance and global offensive r -alliance problems are NP-complete for any fixed r [18]. There are polynomial time algorithms for finding minimum alliances in trees [9, 38, 39]. A polynomial time algorithm for finding minimum defensive alliances in series parallel graphs is presented in [38].

There has also been some work on the parameterized complexity of alliance problems. It is known that DEFENSIVE ALLIANCE and OFFENSIVE ALLIANCE are fixed-parameter tractable when parameterized by the solution size [13, 16]. Alliance problems have been studied with respect to structural graph parameters. In this paper, we show that the problems for all variants are efficiently solvable for much larger graph classes. Kiyomia and Otachi [42] proved that alliance problems can be solved in polynomial time for graphs of bounded clique-width. They also showed that the problems are fixed-parameter tractable when parameterized by the vertex cover number. Ensico [13] showed that the problems of finding minimum size defensive alliances and global defensive alliances are fixed-parameter tractable when parameterized by the combined parameters treewidth and maximum degree. Treewidth [50, 7] is one of the most extensively studied structural parameters in parameterized complexity. It indicates how close a graph is to being a tree. It is particularly interesting because there are many hard problems which become tractable on instances of bounded treewidth. It has also been observed that the problem instances for several practical applications exhibit small treewidth [7, 59]. Hence it is very appealing to obtain FPT algorithms for the DEFENSIVE ALLIANCE and OFFENSIVE ALLIANCE problems using this parameter. Bliem and Woltran [6] proved that defensive alliance problem is W[1]-hard when parameterized by the treewidth of the input graph. It is also known that the OFFENSIVE ALLIANCE problem is W[1]-hard when parameterized by the treewidth of the input graph [22]. This puts these two problems

among the few problems that are FPT when parameterized by solution size but not when parameterized by treewidth (unless $\text{FPT}=\text{W}[1]$).

An alliance S is called a *locally minimal alliance* if for any $v \in S$, $S \setminus \{v\}$ is not an alliance. Bazgan, Fernau and Tuza [1] proved that deciding if a graph contains a locally minimal strong defensive alliance of size at least k is NP-complete, even when restricted to bipartite graphs with average degree less than 3.6; deciding if a graph contains a locally minimal defensive alliance of size at least k , even for bipartite graphs with average degree less than 5.6. The authors also proved that deciding if a graph contains a connected locally minimal strong defensive alliance or a connected locally minimal defensive alliance of size at least k is NP-complete, even when restricted to bipartite graphs with average degree less than $2 + \epsilon$, for any $\epsilon > 0$.

In the first paper on the satisfactory partition problem, Gerber and Kobler [32] considered a generalized version of this problem by introducing weights for the vertices and edges and showed that a general version of the problem is strongly NP-complete. For the unweighted version, they presented some sufficient conditions for the existence of a solution. This problem was further studied in [2, 33, 31]. The SATISFACTORY PARTITION problem is NP-complete and this implies that BALANCED SATISFACTORY PARTITION problem is also NP-complete via a simple reduction in which we add new dummy vertices and dummy edges to the graph [3, 5]. Both problems are solvable in polynomial time for graphs with maximum degree at most 4 [5]. They also studied generalizations and variants of this problem when a partition into $k \geq 3$ nonempty parts is required. Bazgan, Tuza, and Vanderpooten [2, 4] studied an “unweighted” generalization of SATISFACTORY PARTITION, where each vertex v is required to have at least $s(v)$ neighbours in its own part, for a given function s representing the degree of satisfiability. Obviously, when $s = \lceil \frac{d}{2} \rceil$, where d is the degree function, they obtained satisfactory partition. They gave a polynomial-time algorithm for graphs of bounded treewidth which decides if a graph admits a satisfactory partition, and gives such a partition if it exists. Except this result, to the best of our knowledge, the parameterized complexity of the SATISFACTORY PARTITION and BALANCED SATISFACTORY PARTITION problems have not been studied before.

1.6 Graph Theory

In this section, we state some basic definitions in graph theory that are used in this thesis. For standard notations and definitions in graph theory we refer to West [60]. We define a *graph* G to be a finite non-empty set $V = V(G)$ of n vertices together a (multi) set E of m unordered pair of vertices of V . Each pair (u, v) of vertices in E is an *edge* of G ; u and v are said to be *adjacent*. For a graph G , $V(G)$ and $E(G)$ denote the vertex and edge sets of the graph G , respectively. A *loop* in G is an edge of the form $(v, v) \in E(G)$. If an edge (u, v) appears at least twice in E then it is called a *multi-edge*. A graph is *simple* if it has no loops or multi-edges. A graph is a *multi graph* if it contains a self-loop or a multi-edge. A *directed graph*, also called a *digraph*, is a graph in which the edges have a direction. Throughout this thesis we consider undirected simple graphs. A graph H is a *subgraph* of G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$ where of course any edge of in H must have both its endpoints in $V(H)$. If v is a vertex of a graph, then the degree of v , $d(v)$, is the number of edges incident with v . A *walk* in a graph or a multigraph is an alternating sequence of vertices and edges $v_0, e_1, v_1, e_2, \dots, v_{n-1}, e_n, v_n$ beginning and ending with vertices, in which every edge is incident with the vertices immediately preceding and following it. This walk joins v_0 and v_n and may be called a (v_0, v_n) walk. It is closed if $v_0 = v_n$ and *open* otherwise. It is a *path* if all the vertices are distinct. If $v_0 = v_n$ but all other vertices are distinct, it is a *cycle*.

A graph is *connected* if every pair of vertices are joined by a path. If a graph is not connected, it is called *disconnected* and its maximal connected subgraphs are called *components*. A simple and important type of graph is called a tree. A *tree* is a connected graph containing no cycles. A *rooted tree* is a tree in which one vertex has been designated the root. In a rooted tree, the parent of a vertex v is the vertex adjacent to v on the path to the root; every vertex has a unique parent except the root which has no parent. A child of a vertex v is a vertex of which v is the parent. The *height* of a rooted tree is the length of the longest path from the root to any vertex. In this dissertation, the *path length* corresponds to the number of vertices in the path. Consider the tree shown in Figure 1.5. The root of this tree is x_1 ; nodes x_4 and x_5 are children of x_3 . Node x_5 is parent of x_6 . Nodes x_2, x_4, x_6 are leaf nodes. The height of this tree is four. A *forest* is a graph containing no cycles. Some other special types of graphs will also be of interest to us. A graph G is *regular* of degree d if every vertex has degree d . The graph on n vertices in which every vertex is adjacent to

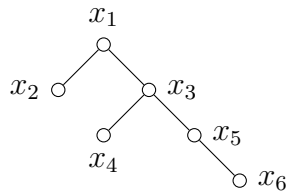


Figure 1.5: A tree of height 4.

every other vertex is called the *complete* graph and denoted by K_n . If the vertices of a graph G can be partitioned into two sets V_1 and V_2 , so that every edge joins a vertex of V_1 to a vertex of V_2 , then G is called *bipartite*. If every vertex of V_1 is adjacent to every vertex of V_2 , then G is *complete bipartite* and denoted by $K_{m,n}$ where $m = |V_1|$ and $n = |V_2|$. *Planar graph* is a graph that can be drawn on the plane so that its edges meet only at vertices. A *clique* is a subset of vertices of an undirected graph such that every two distinct vertices in the clique are adjacent. A *independent set* is a subset of vertices of an undirected graph such that no two vertices in the independent set are adjacent to each other.

A graph G is said to be *chordal* if every cycle of G of length at least 4 has a chord. Figure 1.6 shows a chordal graph. Block graphs form a subclass of chordal graphs that generalize

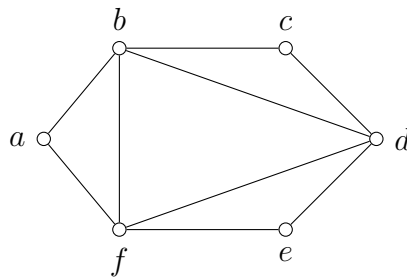


Figure 1.6: A chordal graph

the trees. For any positive integer k , a k -connected component is a maximal k -connected subgraph in G . The 2-connected components of a graph G are called *blocks* of G . Thus a block does not contain any cut-vertex; a *cut-vertex* is a vertex the removal of which would disconnect the remaining graph. Moreover, two blocks of a graph G share at most one vertex of G . A graph is said to be a *block graph* if every block of the graph is a clique. Figure 1.7 shows a block graph.

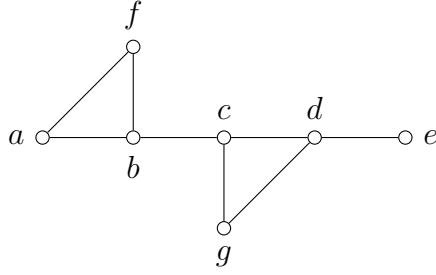


Figure 1.7: A block graph

1.7 Structural Graph Parameters

In this section, we recall definitions of structural graph parameters that are discussed in this thesis. The graph parameters we discuss in this thesis are pathwidth, treewidth, vertex cover number, twin-cover number and feedback vertex set number.

Vertex Cover, Twin-Cover, Feedback Vertex Set

Definition 1.7.1. A set $S \subseteq V(G)$ is a *vertex cover* of $G = (V, E)$ if each edge in E has at least one endpoint in S . The *size* of a smallest vertex cover of G is the *vertex cover number* of G .

We now recall a natural way of generalizing vertex cover to dense graphs. We relax the definition of vertex cover so that not all edges need to be covered.

Definition 1.7.2. An edge is a *twin edge* if its incident vertices have the same closed neighborhood.

Definition 1.7.3. A set $X \subseteq V(G)$ is a *twin-cover* of G if every edge in G is either twin or incident to a vertex in X . We then say that G has *twin-cover number* k if k is the minimum possible size of a twin-cover of G .

An illustration and comparison is provided in Figure 1.8.

Definition 1.7.4. A *feedback vertex set* of a graph G is a set of vertices whose removal leaves G without cycles. The minimum size of a feedback vertex set in G is the *feedback vertex set number* of G , denoted by $\text{fvs}(G)$.

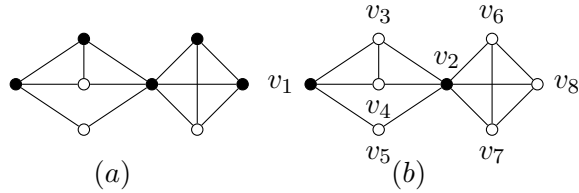


Figure 1.8: (a) A minimum-size vertex cover (size 5 – depicted in black) (b) A minimum-size twin-cover (size 2 – depicted in black).

Neighbourhood Diversity

We say that two (distinct) vertices u and v have the same *neighbourhood type* if they share their respective neighborhoods, that is, when $N(u) \setminus \{v\} = N(v) \setminus \{u\}$. If this is so we say that u and v are *twins*. It is possible to distinguish *true-twins* (those joined by an edge) and *false-twins* (in which case $N(u) = N(v)$).

Definition 1.7.5. [46] A graph $G = (V, E)$ has *neighbourhood diversity* at most k , if there exists a partition of V into at most k sets (we call these sets *type classes*) such that all the vertices in each set have the same neighbourhood type.

If neighbourhood diversity of a graph is bounded by an integer k , then there exists a partition $\{C_1, C_2, \dots, C_k\}$ of $V(G)$ into k type classes. We would like to point out that it is possible to compute the neighborhood diversity of a graph in linear time using fast modular decomposition algorithms [58]. Notice that each type class could either be a clique or an independent set by definition and two type classes are either joined by a complete bipartite graph or no edge between vertices of the two types is present in G . For algorithmic purpose it is often useful to consider a *type graph* H of graph G , where each vertex of H is a type class in G , and two vertices C_i and C_j are adjacent iff there is a complete bipartite clique between these type classes in G . The key property of graphs of bounded neighbourhood diversity is that their type graphs have bounded size. For example, a graph G with neighbourhood diversity four and its corresponding type graph H is illustrated in Figure 1.9.

Pathwidth and Treewidth

We now review the concept of a tree decomposition, introduced by Robertson and Seymour in [50]. Treewidth is a measure of how “tree-like” the graph is.

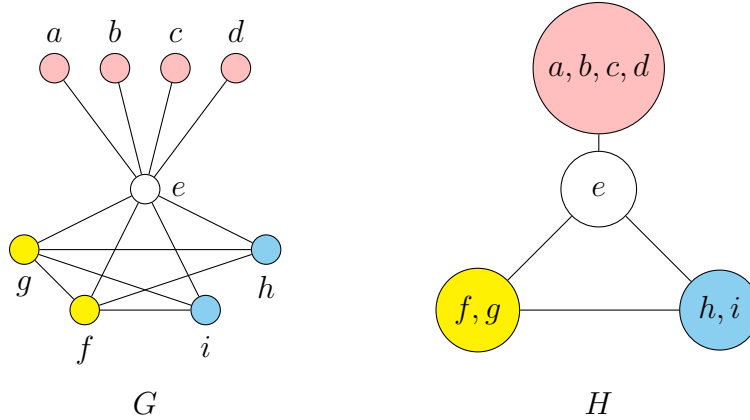


Figure 1.9: A graph G with neighbourhood diversity 4 and its corresponding type graph H .

Definition 1.7.6. [12] A *tree decomposition* of a graph $G = (V, E)$ is a tree T together with a collection of subsets X_t (called *bags*) of V labeled by the nodes t of T such that $\bigcup_{t \in T} X_t = V$ and (1) and (2) below hold:

1. For every edge $(u, v) \in E(G)$, there is some t such that $\{u, v\} \subseteq X_t$.
2. (Interpolation Property) If t is a node on the unique path in T from t_1 to t_2 , then $X_{t_1} \cap X_{t_2} \subseteq X_t$.

Definition 1.7.7. [12] The *width* of a tree decomposition is the maximum value of $|X_t| - 1$ taken over all the nodes t of the tree T of the decomposition. The *treewidth* $tw(G)$ of a graph G is the minimum width among all possible tree decompositions of G .

Example 1. Figure 1.10 gives an example of a tree decomposition of width 2.

A special type of tree decomposition, known as a *nice tree decomposition* was introduced by Kloks [43]. The nodes in such a decomposition can be partitioned into four types (examples in Figure 1.11):

Definition 1.7.8. [11] A tree decomposition is said to be *nice tree decomposition* if the following conditions are satisfied:

1. All bags corresponding to leaves are empty. One of the leaves is considered as root node r . Thus $X_r = \emptyset$ and $X_l = \emptyset$ for each leaf l .

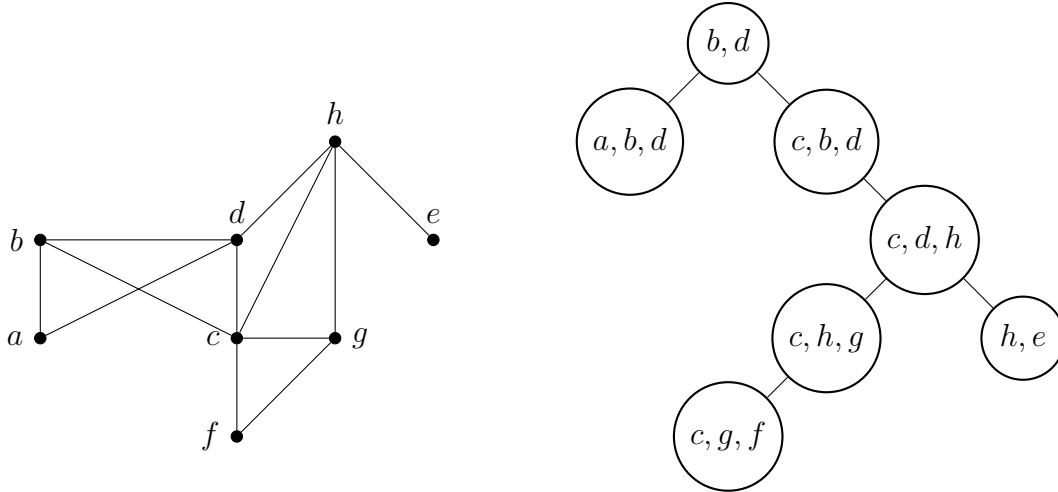


Figure 1.10: Example of a tree decomposition of width 2

2. There are three types of non-leaf nodes:

- **Introduce node:** a node t with exactly one child t' such that $X_t = X_{t'} \cup \{v\}$ for some $v \notin X_{t'}$; we say that v is *introduced* at t .
- **Forget node:** a node t with exactly one child t' such that $X_t = X_{t'} \setminus \{w\}$ for some $w \in X_{t'}$; we say that w is *forgotten* at t .
- **Join node:** a node with two children t_1 and t_2 such that $X_t = X_{t_1} = X_{t_2}$.

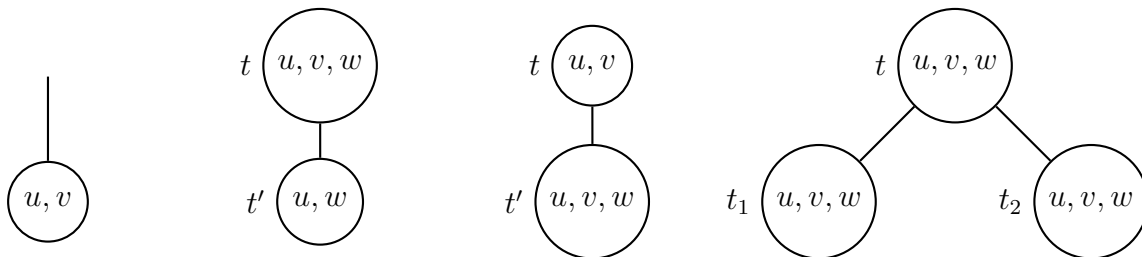


Figure 1.11: The four types of node in a nice tree decomposition. From left to right: a leaf, an introduce node, a forget node, and a join node.

Note that, by the third property of tree decomposition, a vertex $v \in V(G)$ may be introduced several time, but each vertex is forgotten only once. To control introduction of edges, sometimes one more type of node is considered in nice tree decomposition called introduce edge node. An *introduce edge node* is a node t , labeled with edge $uv \in E(G)$, such that $u, v \in X_t$ and $X_t = X_{t'}$, where t' is the only child of t . We say that node t introduces edge

uv . It is known that if a graph G admits a tree decomposition of width at most tw , then it also admits a nice tree decomposition of width at most tw , that has at most $O(n \cdot \text{tw})$ nodes [11].

Definition 1.7.9. If the tree T of a tree decomposition is a path, then we say that the tree decomposition is a *path decomposition*, and use *pathwidth* in place of treewidth.

Example 2. Figure 1.12 gives an example of a path decomposition of width 3.

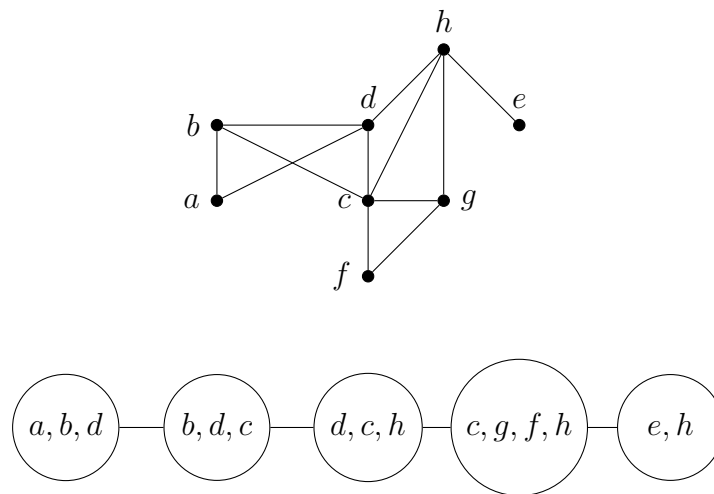


Figure 1.12: Example of a path decomposition of width 3

Treedepth

A rooted forest is a disjoint union of rooted trees. Given a rooted forest F , its *transitive closure* is a graph H in which $V(H)$ 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 .

Definition 1.7.10. The *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 $td(G)$.

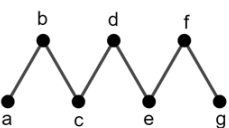
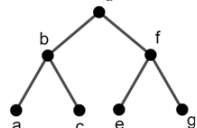
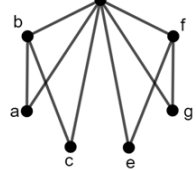
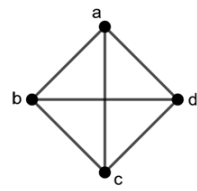

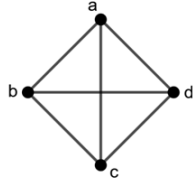
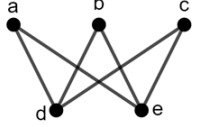
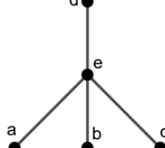
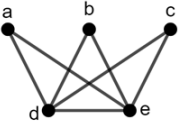
Ex. No.	Graphs	A Rooted Forest of Minimum Depth	Transitive Closure of the Rooted Forest	td
1.				3
2.				4
3.				3

Figure 1.13: Examples explaining treedepth of the graphs

Clique Width

The clique-width of a graph G , denoted by $\text{cw}(G)$, is the minimum number of labels needed to construct G using the following four operations:

1. Create a new graph with a single vertex v with label i (written $i(v)$).
2. Take the disjoint union of two labelled graphs G_1 and G_2 (written $G_1 \cup G_2$).
3. Add an edge between every vertex with label i and every vertex with label j , $i \neq j$ (written η_{ij}).
4. Relabel every vertex with label i to have label j (written $\rho_{i \rightarrow j}$).

We say that a construction of a graph G with the four operations is a c -expression if it uses at most c labels. Thus the clique-width of G is the minimum c for which G has a c -expression.

A c -expression is a rooted binary tree T such that

1. each leaf has label i for some $i \in \{1, \dots, c\}$,

2. each non-leaf node with two children has label \cup , and
3. each non-leaf node with only one child has label $\rho_{i,j}$ or $\eta_{i,j}$ ($i, j \in \{1, \dots, c\}, i \neq j$).

Example 3. Consider the graph P_n , which is simply a path on n vertices. Note that $\text{cw}(P_1) = 1$ and $\text{cw}(P_2) = \text{cw}(P_3) = 2$. Now consider a path on four vertices v_1, v_2, v_3, v_4 , in that order. Then this path can be constructed using the four operations (using only three labels) as follows:

$$\eta_{3,2}(3(v_4) \cup \rho_{3 \rightarrow 2}(\rho_{2 \rightarrow 1}(\eta_{3,2}(3(v_3) \cup \eta_{2,1}(2(v_2) \cup 1(v_1)))))).$$

This construction can readily be generalized to longer paths for $n \geq 5$. It is easy to see that $\text{cw}(P_n) = 3$ for all $n \geq 4$.

A c -expression represents the graph represented by its root. A c -expression of a n -vertex graph G has $O(n)$ vertices. A c -expression of a graph is *irredundant* if for each edge $\{u, v\}$, there is exactly one node $\eta_{i,j}$ that adds the edge between u and v . It is known that a c -expression of a graph can be transformed into an irredundant one with $O(n)$ nodes in linear time. Here we use irredundant c -expression only.

Computing the clique-width and a corresponding c -expression of a graph is NP-hard. For $c \leq 3$, we can compute a c -expression of a graph of clique-width at most c in $O(n^2m)$ time, where n and m are the number of vertices and edges, respectively. For fixed $c \geq 4$, it is not known whether one can compute the clique-width and a corresponding c -expression of a graph in polynomial time. On the other hand, it is known that for any fixed c , one can compute a $(2^{c+1} - 1)$ -expression of a graph of clique-width c in $O(n^3)$ time. For more details see [40].

Relationship Between Graph Parameters

See Figure 1.14 for a schematic representation of the relationship between selected graph parameters. Note that $A \rightarrow B$ means that there exists a function f such that for all graphs, $f(A(G)) \geq B(G)$; therefore the existence of an FPT algorithm parameterized by B implies the existence of an FPT algorithm parameterized by A , and conversely, any negative result parameterized by A implies the same negative result parameterized by B .

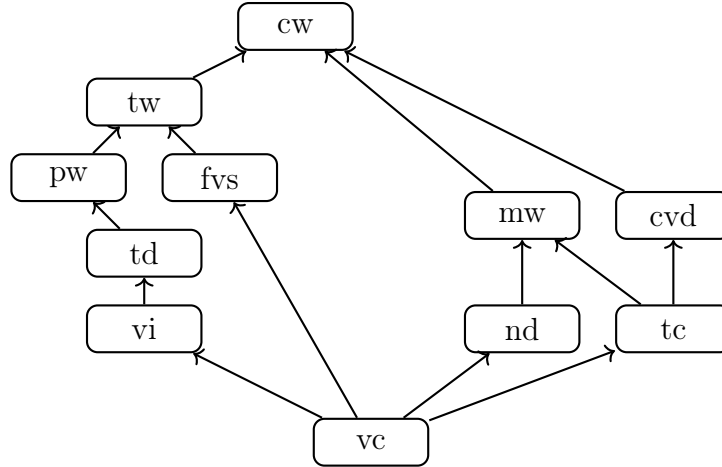


Figure 1.14: Relationship between vertex cover (vc), neighbourhood diversity (nd), twin cover (tc), modular width (mw), cluster vertex deletion number (cvd), feedback vertex set (fvs), pathwidth (pw), treewidth (tw) and clique width (cw). Note that $A \rightarrow B$ means that there exists a function f such that for all graphs G , $f(A(G)) \geq B(G)$.

1.8 An Overview of the Thesis

In Chapter 2, we give a brief introduction to Parameterized Complexity. We formally define parameterized problems, fixed-parameter tractable algorithms and kernelization. Furthermore, we illustrate each of them with some examples. We also give a brief introduction to fixed-parameter intractability and W-hierarchy. Each of these notions are illustrated with examples.

In Chapter 3, we consider the DEFENSIVE ALLIANCE and OFFENSIVE ALLIANCE problems. This chapter is based on the paper [25]. In the DEFENSIVE ALLIANCE problem, given an undirected graph G and a positive integer k , the question is to check whether G has a defensive alliance of size at most k . In the OFFENSIVE ALLIANCE problem, given an undirected graph G and a positive integer k , the question is to check whether G has an offensive alliance of size at most k . We design FPT algorithms for DEFENSIVE ALLIANCE and OFFENSIVE ALLIANCE when parameterized by neighbourhood diversity of the input graph. We prove that OFFENSIVE ALLIANCE is FPT when parameterized by domino treewidth of the input graph. We also prove that DEFENSIVE ALLIANCE and OFFENSIVE ALLIANCE are polynomial time solvable for graphs with bounded treewidth.

In Chapter 4, we study parameterized intractability of the DEFENSIVE ALLIANCE prob-

lem. This chapter is based on the paper [29]. We prove that the DEFENSIVE ALLIANCE problem is $W[1]$ -hard when parameterized by the pathwidth of the input graph. We also prove that the EXACT DEFENSIVE ALLIANCE problem is $W[1]$ -hard when parameterized by the feedback vertex set number and the pathwidth of the input graph. We introduce several variants of DEFENSIVE ALLIANCE that we require in our proofs of above two results. The DEFENSIVE ALLIANCE^F problem generalizes DEFENSIVE ALLIANCE where some vertices are forced to be outside the solution; these vertices are called “forbidden” vertices. Given a graph $G = (V, E)$, a set $V_{\square} \subseteq V(G)$ and an integer $k \in \mathbb{N}$, we study DEFENSIVE ALLIANCE^F, where the goal is to find a defensive alliance $S \subseteq V$ such that (i) $1 \leq |S| \leq k$, and (ii) $S \cap V_{\square} = \emptyset$. The DEFENSIVE ALLIANCE^{FN} problem is a further generalization that, in addition, requires some “necessary” vertices to be in S . Given a graph $G = (V, E)$, a set $V_{\square} \subseteq V(G)$, a set $V_{\Delta} \subseteq V$, and an integer $k \in \mathbb{N}$, we study DEFENSIVE ALLIANCE^{FN}, where the goal is to find a defensive alliance $S \subseteq V$ such that (i) $1 \leq |S| \leq k$, (ii) $S \cap V_{\square} = \emptyset$, and (iii) $V_{\Delta} \subseteq S$. We prove that the DEFENSIVE ALLIANCE^{FN} problem is $W[1]$ -hard when parameterized by the size of a vertex deletion set into trees of height at most 4. We also prove that the DEFENSIVE ALLIANCE^F problem is $W[1]$ -hard when parameterized by the pathwidth of the graph.

In Chapter 5, we consider the problem LOCALLY MINIMAL DEFENSIVE ALLIANCE, which takes as input an undirected graph G and a positive integer k , and the objective is to decide if G has a locally minimal defensive alliance of size at least k . This chapter is based on the paper [26] and the manuscript [27]. We design a polynomial-time algorithm for the CONNECTED LOCALLY MINIMAL STRONG DEFENSIVE ALLIANCE on trees. We prove that LOCALLY MINIMAL DEFENSIVE ALLIANCE problem is NP-complete, even when restricted to planar graphs. We give a randomized FPT algorithm for the EXACT CONNECTED LOCALLY MINIMAL DEFENSIVE ALLIANCE problem using color coding technique. We give an FPT algorithm for LOCALLY MINIMAL DEFENSIVE ALLIANCE when parameterized by neighbourhood diversity of the input graph. We prove that EXACT CONNECTED LOCALLY MINIMAL DEFENSIVE ALLIANCE parameterized by treewidth is $W[1]$ -hard and thus not FPT (unless $FPT=W[1]$). Finally we show that the LOCALLY MINIMAL DEFENSIVE ALLIANCE problem is polynomial time solvable for graphs with bounded treewidth.

In Chapter 6, we consider the SATISFACTORY PARTITION and BALANCED SATISFACTORY PARTITION problems. In the SATISFACTORY PARTITION problem, given a graph $G = (V, E)$, the goal is to find a nontrivial partition (V_1, V_2) of V such that for every $v \in V$,

if $v \in V_i$ then $d_{V_i}(v) \geq d_{V_{3-i}}(v)$. In the BALANCED SATISFACTORY PARTITION problem, given a graph $G = (V, E)$, the goal is to find a nontrivial partition (V_1, V_2) of V such that $|V_1| = |V_2|$ and for every $v \in V$, if $v \in V_i$ then $d_{V_i}(v) \geq d_{V_{3-i}}(v)$. This chapter is based on the papers [28, 23, 24]. We design a polynomial-time algorithm for the SATISFACTORY PARTITION problem for block graphs. We prove that the SATISFACTORY PARTITION and BALANCED SATISFACTORY PARTITION problems are fixed parameter tractable (FPT) when parameterized by neighbourhood diversity. We show that the SATISFACTORY PARTITION and BALANCED SATISFACTORY PARTITION problems can be solved in polynomial time for graphs of bounded clique-width. Finally we prove that the BALANCED SATISFACTORY PARTITION problem is W[1]-hard when parameterized by treewidth.

Chapter 2

Introduction to Parameterized Complexity

The time taken by an algorithm is a central measure of its efficiency. For many problems, there are algorithms whose time complexity is polynomial in the size of the instance. However, there are a multitude of problems for which no polynomial time algorithm is known to exist. The known algorithms for such problems usually tend to be *exponential* in the size n of the instance. As a result, the time complexity grows a lot faster compared to n , making the problem computationally intractable even for relatively smaller instances.

2.1 Fixed Parameter Tractability

A popular method to soften this exponential blow is to look at parameterized problems. Here, every instance is attached with a parameter k , thereby expanding the set of all instances to $I \times \mathbb{N}$. Any element of this set is of the form (x, k) where x is the instance, and k is the value of some parameter associated to x . A parameterized problem now asks a Yes/No question, often linking x to k , and is characterised by a subset $P \subseteq I \times \mathbb{N}$ of yes instances.

One wonders then, as to how adding a parameter can help in designing more efficient algorithms to the exponential problems above. The key idea is to capture and restrict the exponential growth in running times to the parameter, thereby obtaining an algorithm

which takes time possibly exponential in the parameter k , but polynomial in the size n of the instance x . Such a tactic makes the problem tractable over the subset of instances having a parameter less than a fixed small maximum value of k . This happens because the exponential term in k gets bounded above by a constant, thereby rendering an algorithm with time complexity polynomial in n . Such parameterized problems are said to be fixed-parameter tractable (FPT). Formally,

Definition 2.1.1. A parametrized problem $P \subseteq I \times \mathbb{N}$ is said to be *fixed-parameter tractable (FPT)* if it is possible to determine accurately, for input $(x, k) \in I \times \mathbb{N}$, if $(x, k) \in P$ in time $f(k) \cdot n^c$ where f is some (usually computable) function, n is the size of the main part of the input x , that is $|x| = n$ and c is a constant independent of k . The algorithm that performs this task is called an *FPT algorithm*.

We now look at some techniques that are used in constructing FPT algorithms, and see some examples of these algorithms in graph theory.

2.1.1 Kernelization

Imagine that you are searching for a 5-letter across word while solving a crossword (or perhaps a Wordle). A general brute-force algorithm would expect you to think of all possible 5-letter words one at a time and match them with the given meaning or clue (at least 1,50,000 meaningful ones!). Now suppose that the first and third letter of this word are known to be 'A' via solved down clues. This now reduces your search to only those 5-letter words of the form A?A?? (around 18 meaningful words of this form). So, obtaining some trivial information about the word, using the down clues, helped restrict our search to a much smaller set, and with variations over those positions in the word that did not intersect with any down clues. In other words, we restricted ourselves to the core letters of the word.

Similarly, say now you are solving an MCQ with 5 options and do not know the answer to the question. If you make a guess, the chances of getting a correct answer is just 0.2. But suppose, now you were able to trivially eliminate three irrelevant and incorrect options out of the five. Then your problem gets restricted to choosing one of the 2 options, with a 0.5 chance of getting it correct. You have successfully reduced your options by reaching the core

two options of the MCQ. Even while finding possible solutions to a given Sudoku, finding some numbers by hand with the help of the basic rules, reduces your search for solutions, thereby allowing for faster brute-force or branching methods.

The above scenarios capture the essence of kernelization techniques: To make some polynomial time deductions on the given instance, which restricts the algorithm to the computationally hard ‘core’ (kernel) of the problem, thereby reducing the time complexity. Formally, we define kernelization as follows:

Definition 2.1.2. A *kernelization algorithm* takes as input an instance $(x, k) \subseteq I \times \mathbb{N}$ and returns in polynomial time, an equivalent instance (x', k') , such that $|x'| \leq g(k)$ and $k' \leq g(k)$ for some computable function $g(k)$.

Two instances (x, k) and (x', k') are said to be *equivalent instances* of P when $(x, k) \in P$ if and only if $(x', k') \in P$. Note that once $k' \leq g(k)$, there might be no further reduction in the value of the parameter, and thus repetitive applications of a kernelization algorithm need not reduce the problem to triviality. However, in most cases, a repetitive and exhaustive application of kernelization algorithms provide an equivalent instance with size bounded by a function of k , thereby paving way for an FPT algorithm.

Vertex Cover

We shall look at a kernelisation algorithm for finding a smallest vertex cover of a graph $G = (V, E)$. We say that the vertices u and v *cover* the edge (u, v) . A *vertex cover* of G is a subset $S \subset V$ such that every edge in E is covered by some element of S . This example is taken from [11]. We define the problem as follows:

VERTEX COVER

Input: A graph $G = (V, E)$ and a positive integer k .

Question: Does G have a vertex cover of size at most k ?

We shall find a simple kernelization algorithm for this problem parameterized by the solution size k . Recall from graph theory that, a vertex v is said to be a neighbour of u if (u, v) is

an edge. The set of all neighbours of a vertex v is denoted by $N(v)$, and the *degree* $d(v)$ is defined to be the cardinality $|N(v)|$.

Let us look at the vertices with degree 0 (isolated vertices). If v is an isolated vertex, then it does not cover any edge. Thus, v need not be a part of any vertex cover for the graph G , and hence can be ignored. Note that ignoring this vertex does not affect the size of the vertex cover, and thus keeps the parameter the same. This gives us our first reduction rule for kernelization:

Reduction VC 1. If v is an isolated vertex in G , delete v from G . The new instance is $(G - v, k)$

Note that the above rule satisfies all conditions of a kernelization algorithm. Using this rule we can eliminate all isolated vertices. Let us move on to the set of one-degree vertices. Let v be a vertex with $d(v) = 1$, and w be its unique neighbour. If S is a vertex cover of size k , then either $v \in S$ or $v \notin S$. If $v \notin S$, then w must lie in S so as to cover the edge (v, w) . Alternatively, if $v \in S$, then v covers exactly one edge $e = (v, w)$ while w covers more edges than v does, and also covers the edge e . So, replacing v with w results in a more optimal covering of edges. Thus, we can always find a smallest vertex cover which has exactly one vertex out of v and w , preferably w .

So, if v is a vertex with $d(v) = 1$, then we have obtained a sure element w , the only neighbour of v , in any optimal vertex cover, allowing us to forget both the vertices v and w from the graph G , and reduce the parameter size by 1. This gives us the next reduction rule:

Reduction VC 2. If v is a vertex of degree one and w is the neighbour of v in G , then delete w from G and decrease the parameter k by 1. The new instance is $(G - w, k - 1)$

Now, we focus our attention to vertices with higher degree. Suppose we are searching for a vertex cover S of size at most k , and v is a vertex with degree $d(v) > k$, then v must belong to S . As v has at least $k + 1$ edges in G and if $v \notin S$, then all these $k + 1$ edges will have to be covered by $k + 1$ distinct vertices in S . But this is not possible as $|S| \leq k$. This idea gives the next reduction rule:

Reduction VC 3. If v is a vertex of degree at least $k + 1$, then delete v from G , and decrease the parameter k by 1. The new instance is $(G - v, k - 1)$.

These three rules are sufficient to demonstrate a simple and effective kernelisation algorithm. Together, they attack on all vertices with degrees $d(v) \leq 1$ or $d(v) \geq k + 1$. We apply these rules exhaustively, that is, until it is not possible to apply any of the rules in the output instance. It is important to note that the application of the three rules can be exhausted in finite time as each rule reduces the size of the input graph by at least 1 vertex.

Similar to most kernelization algorithms, an exhaustive application of these rules returns an equivalent instance (G', k') , called the *kernel*, such that the size of $V(G')$ is bounded by some function of k . We shall now prove this fact:

Theorem 2.1.1. Let (G', k') be the new instance obtained from (G, k) such that none of the reduction rules $VC.1, VC.2, VC.3$ is applicable to G' . If (G', k') is a yes instance then G' has at most k^2 vertices and at most k^2 edges.

Proof. Let (G', k') be obtained from (G, k) after an exhaustive applications of the three rules. Suppose that S' is a vertex cover of G' of size $\leq k'$, that is, (G', k') is an yes-instance. Note that the application of any of the three rules does not cause the value of k to increase. Thus, $|S'| \leq k' \leq k$. We count first, the edges in G' . As every edge in G' is covered by a vertex in S' ,

$$|E(G')| \leq \Delta(G') \cdot |S'|$$

where $\Delta(G')$ is the maximum degree of G' , which is less than k due to exhaustion application of Reduction VC 3. So,

$$|E(G')| \leq \Delta(G') \cdot |S'| \leq k \cdot k \leq k^2.$$

Thus, G' has at most k^2 edges. Now, for the vertices, by the first handshake lemma in graph theory, we have:

$$2|E(G')| = \sum_{v \in V(G')} d(v)$$

We know by exhaustive application of Reduction VC.1 and VC.2, that $d(v) \geq 2$. Further, we saw that $|E(G')| \leq k^2$. Combining both these facts in the previous equation, we get

$$2k^2 \geq 2|E(G')| = \sum_{v \in V(G')} d(v) \geq \sum_{v \in V(G')} 2 = 2|V(G')|$$

or, $|V(G')| \leq k^2$, as required. □

The above theorem has the following implications. Given a graph G with n vertices, we can find a kernel (G', k') in time polynomial in n , say $p(n)$. Now, we can check if the number of edges or vertices in G' is more than k^2 in time $\mathcal{O}(k^2)$. If either number has size greater than k^2 , we declare (G, k) to be a no-instance by the contrapositive of Theorem 2.1.1. If this is not the case, then we search for a vertex cover by some brute-force approach. We look at all possible choices of a vertex cover of size k' and check for each choice, if it covers each of the k^2 edges or not. This takes $g(k) = \mathcal{O}(k^{2k+2})$ time, exponential in k . If we find a vertex cover of size k' , we say that (G, k) is a yes-instance, else it would be a no-instance. This process determines whether (G, k) is a yes-instance or not in $p(n) + g(k)\mathcal{O}(k^2)$ time, thereby giving us an FPT algorithm.

What we demonstrated above was the formation of a kernelisation algorithm by observing simple facts. More efficient kernelisation algorithms give a kernel of smaller sizes, and can be obtained via a deeper study of the structural aspects of the given instance. There are better kernelisation techniques which use structural features, like crown decompositions, sunflower lemma, or expansion lemma, which can provide smaller kernels. We end this section by mentioning a fascinating connection between FPT and kernelization algorithms, which follows easily from the definitions:

Theorem 2.1.2. [11] A parameterized problem admits a kernelization algorithm if and only if it is fixed-parameter tractable.

2.1.2 Bounded Search Trees

The vertex cover algorithm given in Section 2.1.1 takes about $p(n) + \mathcal{O}(k^{2k+4})$ running time. Even though the exponential time has been captured only in k , it is still very high and we can try to improve it by finding a better algorithm.

The key idea now is to make use of *branching*: at every step, we take the given instance and break it into some s disjoint and exhaustive cases, with each case working on some reduced simpler instance. What we require is that each of these simpler instances has a solution that can be extended to a solution for the parent instance, and at least one of these extended solutions must match with an optimal solution on the parent instance as desired by the problem. The branching is used recursively at every simpler instance, forming some

sort of recursive search trees.

The recursive search will have a leaf instance which is trivial to solve. Such instances need not be branched further, and depict the end of the recursion tree. The key idea is that if the number of nodes in the recursion tree is bounded by a function of the parameter k , say $f(k)$, and takes polynomial $p(n)$ time at each node, we get a *Bounded Search Tree* which solves the problem in $\mathcal{O}(f(k)q(n))$ time, thereby giving perhaps an efficient FPT algorithm. The number of nodes depend on the number of branches s at each level and the depth of the tree d , and effectively have to be kept small. Let us use these ideas to give a better FPT algorithm for the vertex cover problem.

Vertex Cover

This example is taken from [11]. Suppose G is a graph and S is a vertex cover for G , then the key idea is to note that for every vertex v , either v or $N(v)$ must be a part of S so as to cover all the $d(v)$ edges adjacent to v . This will be our criteria for branching. The branching stops when we get a graph with all vertices having $d(v) \leq 1$, as the vertex covers for such graphs can be found easily in polynomial time, or if the parameter value $k \leq 1$, where it becomes easier to determine if a vertex cover of required size will exist or not. The algorithm runs as follows: at every instance (G, k) , we pick a vertex v with the highest degree. Note that v can be found in $\mathcal{O}(n^2)$ time. We branch the instance into 2 cases - either $v \in S$ or $N(v) \subset S$. The first case returns the instance $(G - v, k - 1)$, and the second case returns the instance $(G - N(v), k - |N(v)|)$.

We must now calculate the number of nodes in this recursion tree. Note that, as the tree has 2 branches at every node, depth d , and has $l = 2^d$ leaves, then the total number of nodes in the tree equals $\sum_{i=0}^d 2^i = 2^{d+1} - 1 = 2l - 1$. Thus, our tree has $\mathcal{O}(l)$ nodes and so, it suffices to estimate the size of l . Let $L(k)$ be the number of leaves in a tree starting at parameter k . We saw that $k \leq 1$ is a trivial case and itself forms a leaf. so, $L(0) = L(1) = 1$. Otherwise, for $k > 1$, we branch as explained above. The first branch then has $L(k - 1)$ leaves and the second branch has at most $L(k - 2)$ leaves. As $L(k)$ is increasing in k and the parameter reduces by at least 2 in the second branch as there exists a vertex v of degree $d(v) \geq 2$ in a non trivial instance. The leaves needed are the sum of leaves present in the sub-trees. Thus, $L(k) \leq L(k - 1) + L(k - 2)$. Considering the worst case scenario, we get

the recursion:

$$L(i) = \begin{cases} L(i-1) + L(i-2) & \text{if } i \geq 2, \\ 1 & \text{if } i \leq 1 \end{cases}$$

Observe that the sequence $L(i)$ is the Fibonacci sequence. Such recurrence relations are solved by substituting $L(i) = x^i$ and solving for x . Making the substitution gives us $x^i = x^{i-1} + x^{i-2}$, or $x^2 - x - 1 = 0$, giving a quadratic equation with the solutions $x = \phi = 1.618$ (the Golden ratio) and $x = -0.618$. So, the solution to the recurrence relation will be of the form $L(i) = c_1 1.618^i + c_2 (-0.618)^i$ for some $c_1, c_2 \in R$. We can determine c_1 and c_2 by substituting the initial conditions $L(0) = L(1) = 1$. We get

$$c_1 + c_2 = 1 \quad \text{and} \quad 1.618c_1 - 0.618c_2 = 1.$$

We get $c_1 = 1.618/\sqrt{5}$ and $c_2 = -0.618/\sqrt{5}$. So we have

$$L(k) = \frac{1.618^{k+1}}{\sqrt{5}} + \frac{-0.618^{k+1}}{\sqrt{5}} \leq \frac{2}{\sqrt{5}} \times 1.618^{k+1}.$$

Thus the number of nodes in the search tree is $\mathcal{O}(1.618^k)$, and the algorithm takes $\mathcal{O}(1.618^k n^2)$ time to find a vertex cover in (G, k) . Substituting these values with our kernelization algorithm, gives us an improved FPT algorithm with time $p(n) + \mathcal{O}(1.618^k k^4)$, thereby concluding our discussion on bounded search trees.

2.1.3 FPT via Integer Linear Programming (ILP)

Given two parametrized problems A and B , complexity theory provides us with the means to reduce an instance of problem A to an equivalent instance of problem B . Then, if an algorithm is known for problem B , we can construct an algorithm for problem A by combining the reduction algorithm with the algorithm for problem B . If further, the reduction happens in polynomial time and B has an FPT algorithm, the above process gives an FPT algorithm for problem A as well. In this section, we shall let B be the classical *Integer Linear Programming problem*, and show how FPT algorithms can be designed for other problems,

via the ILP method.

The ILP Problem: Integer linear programming (ILP) is defined as follows:

p-ILP
Input: A matrix $A \in \mathbb{Z}^{m \times p}$, and vectors $b \in \mathbb{Z}^p$ and $c \in \mathbb{Z}^p$.
Question: Find a vector $x \in \mathbb{Z}^p$ that satisfies the m inequalities, that is, $A \cdot x \geq b$.
Parameter: p , the number of variables.

Lenstra [47] showed that *p*-ILP is FPT with running time doubly exponential in p , where p is the number of variables. Later, Kannan [41] proved an algorithm for *p*-ILP running in time $p^{O(p)}$. In our algorithms, we need the optimization version of *p*-ILP rather than the feasibility version. We state the minimization version of *p*-ILP as presented by Fellows et. al. [14]. The problem and the result are formally stated as follows.

p-OPT-ILP
Input: A matrix $A \in \mathbb{Z}^{m \times p}$, and vectors $b \in \mathbb{Z}^p$ and $c \in \mathbb{Z}^p$.
Question: Find a vector $x \in \mathbb{Z}^p$ that minimizes the objective function $c^T \cdot x$ and satisfies the m inequalities, that is, $A \cdot x \geq b$.
Parameter: p , the number of variables.

Lemma 1. [14] *p*-OPT-ILP can be solved using $O(p^{2.5p+o(p)} \cdot L \cdot \log(MN))$ arithmetic operations and space polynomial in L . Here L is the number of bits in the input, N is the maximum absolute value any variable can take, and M is an upper bound on the absolute value of the minimum taken by the objective function.

Given any problem P with parameter k , we must try to reformulate it as an ILP problem with at most $f(k)$ variables for some computable function f in order to apply the ILP technique. This is the most crucial requirement for this technique. If then, substituting L , M and N gives a running time polynomial in n , we shall obtain an FPT algorithm for the problem P .

Imbalance

We shall look at the example of the IMBALANCE problem. This example is taken from [15]. Suppose we order the vertices of a graph $G = (V, E)$ in a straight line, and call this ordering π . For any vertex v , we let $L_\pi(v)$ ($R_\pi(v)$) be the set of all neighbours of v to the left (right) of v in the ordering π . The *imbalance* $i(v)$ of the vertex v is defined to be the difference between the cardinalities of the two sets $L_\pi(v)$ and $R_\pi(v)$, and the imbalance of the graph G is defined as $i(G) = \sum_{v \in V(G)} i(v)$. The problem is to find an ordering of the vertices of G that minimises the imbalance $i(G)$.

We shall attempt to reformulate this as an ILP problem. Let S be the minimum vertex cover of the graph G , with $|S| = s$ vertices, and let π_S be an ordering of S . The aim would be to infuse the vertices of $V \setminus S$ into the fixed ordering π_S in such a way so as to minimise $i(G)$. This process will be repeated for every $s!$ orderings of the vertex cover S , to get the minimum possible imbalance. So, let us fix an ordering π_S with the order (v_1, v_2, \dots, v_s) and let $I = V \setminus S$. Note that I is an independent set.

Let I_i be those vertices of I which will be infused between v_i and v_{i+1} in π_S for $i \in \{1, 2, \dots, s-1\}$ and I_0 are those vertices infused before v_0 . Thus the set $\{I_i : 0 \leq i \leq s\}$ partitions I . Note that as I is an independent set, that is, the vertices of I_i are not adjacent to each other, for any fixed i . Thus, a change in order of vertices within I_i does not affect the imbalance $i(G)$. Hence, it suffices to find an optimal partition of I into $\{I_i\}$.

Constraint 1: Let $S' \subset S$, then define $I(S') = \{x \in I : N(x) = S'\}$. Define the variables $x_{S'}^i = |I_i \cap I(S')|$ for all $S' \subset S$ and $i = 1, 2, \dots, s$ to be used in the ILP. Then, as for a fixed S' , $\{I_i \cap I(S')\}$ partitions $I(S')$. We have the following constraint:

$$\sum_{i=0}^s x_{S'}^i = |I(S')| \quad \forall S' \subset S$$

Constraint 2: Now, we shall constraint the imbalance of the vertices in S . Let x_i be the imbalance of the vertex $v_i \in S$. This will also be a variable in our ILP formulation. Let e_i be the, roughly, the imbalance in v_i calculated only using the vertices of S and the ordering π_S ,

or more precisely, $e_i = L_{\pi_S}(v_i) - R_{\pi_S}(v_i)$, without the modulus sign. Then, the imbalance of v_i is given by:

$$x_i = \left\| e_i + \sum_{\{S' \subset S : v_i \in S'\}} \left(\sum_{j=0}^{i-1} x_{S'}^j - \sum_{j=i}^s x_{S'}^j \right) \right\|.$$

Note that there is a modulus sign in this constraint which makes the constraint non-linear. However, we can split the problem into 2^s separate ILP problems by multiplying the RHS of the above constraint by either $t_i = 1$ or $t_i = -1$, independently for each i . This completes constraint 2.

We finally add the constraint that all $x_{S'}^i$ and x_i must be nonnegative quantities to complete our ILP formulation. For each $S' \subseteq S$, we partition $I(S')$ into s sets $I(S') \cap I_i$, $i = 0, 1, \dots, s$. It may be noted that for each i , every vertex in $I(S') \cap I_i$ will have the same imbalance; this imbalance depends only on the ordering of elements of S' in π_S and can be calculated easily. This imbalance is stored in $z_{S'}^i$. As $x_{S'}^i$ is the number of vertices which have imbalance $z_{S'}^i$, we add the second term in the objective function to count total imbalances due to the vertices in I . Therefore the vertices in $I(S') \cap I_i$ contribute $z_{S'}^i x_{S'}^i$ imbalance to the total imbalance $i(G)$.

Thus, given an ordering π_S of the vertices of S , we have 2^s ILPs of the form:

$$\text{Minimize } i(G) = \sum_{i=1}^s t_i \cdot x_i + \sum_{i=0}^s \sum_{S' \subset S} z_{S'}^i x_{S'}^i$$

subject to

$$\begin{aligned} \sum_{i=0}^s x_{S'}^i &= |I(S')| \quad \forall S' \subset S \\ x_i &= t_i e_i + \sum_{\{S' \subset S : v_i \in S'\}} \left(\sum_{j=0}^{i-1} t_i x_{S'}^j - \sum_{j=i}^s t_i x_{S'}^j \right), \quad i \in \{1, \dots, s\} \\ x_i, x_{S'}^i &\geq 0 \quad \forall S' \subset S, i \in \{0, 1, \dots, s\} \end{aligned}$$

There are a total of $s + s2^s$ variables in the formulated ILP. In order to satisfy the requirement that the number of variables in the ILP problem must be bounded by a function of the parameter, we shall parameterize the IMBALANCE problem with the vertex cover number s

of the input graph G . There are a total of $s!2^s$ ILPs of the above type. For each ILP, the input has size $L = \mathcal{O}(n^2)$ and both M, N are bounded by n . So, each ILP takes $\mathcal{O}(g(s)^{g(s)}n^2\log(n^2))$ time, with $g(s) = s(1 + 2^s)$, thus giving an FPT algorithm for the IMBALANCE problem parametrised by the vertex cover number s which runs in $\mathcal{O}(s!2^s g(s)^{g(s)}n^2\log(n^2))$ time.

This concludes our brief exposition on some of the techniques used to find fixed-parameter tractable algorithms.

2.2 Fixed-Parameter Intractability

Problems in the set P , those which have polynomial time algorithms, are computationally the easiest problems to solve. For those not in P , we saw that FPT-algorithms gave some relief by making these harder problems computationally tractable under the scenario of a bounded parameter k . However, presently, not all problems have known FPT-algorithms. Evidently, these problems become the *hardest* ones to solve, and gain special attention in the theory of parameterized complexity, thereby forming the *W-classes of complexity*. We shall discuss these concepts briefly in this section.

Let us first understand the concept of *hardness* between problems via the theory of classical complexity. Suppose we are given two decision problems P, Q and an algorithm \mathcal{A} , which can convert an instance of P to an equivalent instance of Q in polynomial time. Now, if Q has a polynomial time solution, then P can also be solved in polynomial time: First convert the instance of P to an equivalent instance of Q using the algorithm \mathcal{A} and then solve the problem Q , each takes polynomial time. However, given a polynomial time algorithm for P , it is not necessary that a polynomial time algorithm exists for Q . Stated otherwise, just finding a polynomial time algorithm for P is not enough to find a polynomial time algorithm for Q , and so, we say that Q is *polynomially harder* to solve than P .

More formally, the algorithm $\mathcal{A} : P \xrightarrow{P} Q$ is said to be a *polynomial time reduction* from P to Q . If such an algorithm \mathcal{A} exists, then we say that Q is *polynomially harder* than P , or $P \leq_P Q$. We say that the problems P and Q are *polynomially equivalent* if $P \leq_P Q$ and $Q \leq_P P$, denoted by $P =_P Q$.

This idea of polynomial hardness helps study the set of problems not in P , thereby

characterising polynomial intractability. For example, the NP-complete problems are the polynomially hardest problems in the class NP, and are all polynomially intractable, making them a special focus in the theory of classical complexity. Taking inspiration from these ideas, we can try to study fixed-parameter intractability by defining the concepts of *parameterised reductions* or *parameterised hardness*, and then proceeding in a way similar to the classical theory of complexity.

2.2.1 Parameterized Reduction

We begin by defining the concepts of parameterised reductions in a manner similar to polynomial reductions. To begin with, we would like our parameterised reduction to be an algorithm \mathcal{A} that reduces an instance of P to an equivalent instance of Q in *FPT* time (time complexity of the form $f(k)n^c$), for *parameterised problems* P and Q . However, unlike the polynomial case, this is not strong enough to suit our requirements. Note that, the crucial and desired implication of a polynomial reduction from P to Q , is that a polynomial algorithm for Q will generate a polynomial algorithm for P as well. Likewise, a parameterised reduction from P to Q must be such that the existence of an FPT algorithm for Q must imply the existence of an FPT algorithm for P . Just the above definition is not sufficient to guarantee this implication, and thus we add another clause while defining parameterised reductions as follows:

Definition 2.2.1. [12] Given two parameterised problems P and Q , an algorithm \mathcal{A} is said to be a *parameterised reduction* from P to Q , if for every instance (x, k) of P , the algorithm returns an equivalent instance (x', k') of Q in an FPT running time $f(k) \cdot |x|^c$, such that $k' \leq g(k)$ for some computable functions $f(k)$, $g(k)$ and a constant c .

Note that the instances (x, k) of P and (x', k') of Q are said to be *equivalent* when (x, k) is a yes instance of P if and only if (x', k') is a yes instance of Q . Further, we can, without loss of generality, assume that $f(k)$ and $g(k)$ are increasing functions.

We shall now prove that the above definition is sufficient for our requirements:

Lemma 2. [11] *If there is a parameterized reduction from P to Q and Q admits an FPT algorithm, then P has an FPT algorithm as well.*

Proof. Suppose (x, k) is a given instance of problem P . We have to find an FPT algorithm to determine whether (x, k) is a yes instance or not. The idea would be to use the parameterised reduction algorithm \mathcal{A} to obtain an equivalent instance (x', k') for Q , and then solve it using the known FPT algorithm for Q . We must show that this process takes FPT time for P . First, observe that obtaining the equivalent instance (x', k') via the algorithm \mathcal{A} takes time $T_1 = f(k)|x|^c \leq f(k)|x|^{cd}$ where c and d are constants.

Further, as Q has an FPT algorithm, solving the instance (x', k') would take time $T_2 = h(k')|x'|^d$. Here, by replacing $h(k)$ with $\min \{h(i) : 1 \leq i \leq k\}$, we can without loss of generality assume that $h(k)$ is a non-decreasing function. Now, as $k' \leq g(k)$ and $h(k')$ is non-decreasing, we have $h(k') \leq h \circ g(k)$. Further, the size $|x'|$ is bounded above by the running time of algorithm \mathcal{A} , we have $|x'| \leq f(k)|x|^c$. Combining both these facts, we get the bound $T_2 \leq h \circ g(k)f(k)^d \cdot |x|^{cd}$.

Finally, the time taken for this process,

$$T = T_1 + T_2 \leq [f(k) + h \circ g(k)f(k)^d] \cdot |x|^{cd}$$

Thus, showing that the algorithm for P takes FPT time. □

Based on the definition of a parameterised reduction and the above lemma, we can call a problem Q to be *FPT-harder* than another problem P if there is a parameterised reduction algorithm \mathcal{A} from P to Q . This is denoted by $P \leq_{FPT} Q$. Further, problems P and Q would be called *FPT-equivalent*, or $P =_{FPT} Q$, if $P \leq_{FPT} Q$ and $Q \leq_{FPT} P$. Let us look at examples of some known parameterized reductions showing that CLIQUE can be reduced to some basic problems:

Theorem 2.2.1. [11] *There is a parameterized reduction from CLIQUE to INDEPENDENT SET.*

Proof. Let (G, k) be an instance of CLIQUE. We construct an instance (G', k') of INDEPENDENT SET as follows. We let $V(G') = V(G)$ and $E(G') = \{(u, v) \mid (u, v) \notin E(G), u \neq v\}$ and $k' = k$. Clearly, G' can be constructed in polynomial time and G has a clique of size k if and only if G' has an independent set of size k . □

The MULTICOLOURED CLIQUE problem takes as input, a graph G , an integer k , and a

partition (V_1, V_2, \dots, V_k) of the vertices of G . The problem is to decide whether G has a k -clique having exactly one vertex from each set V_i .

Theorem 2.2.2. [11] *There is a parameterized reduction from CLIQUE to MULTICOLOURED CLIQUE.*

Proof. Let (G, k) be an instance of CLIQUE. We construct an instance $(G', k, (V_1, V_2, \dots, V_k))$ of MULTICOLOURED CLIQUE as follows. Begin constructing G' by making k -copies of the graph G . Let $V(G) = \{v_1, v_2, \dots, v_n\}$ be the vertices of the original graph. We can then, label the vertices of the k -copies of G by $V_i = \{v_j^i \mid j \in [n]\}$ for all $i \in [k]$.

Next, we shall add additional edges as follows: for $v_l^i \in V_i$ and $v_t^j \in V_j$, we add the edge (v_l^i, v_t^j) to $E(G')$ if and only if $(v_l, v_t) \in E(G)$, $l \neq t$. Note that construction of G' takes FPT time. Further, it can be proven that G has a clique if and only if G' has an isomorphic multicoloured clique. Thus $k' = k$, with $g(k) = 1$. \square

It is easy to see from the definitions that a parameterised reduction need not be a polynomial time reduction, for the reduction algorithm itself may not run in polynomial time. However, quite interestingly, every polynomial reduction need not be a parameterised reduction as well! Take for example, the reduction of VERTEX COVER to INDEPENDENT SET. From graph theory, we know that, S is a vertex cover of size at most k for graph G with vertices $V(G)$, if and only if the set $V(G) \setminus S$ is an independent set of size at least $n - k$. We then, have a polynomial reduction which, given a graph (G, k) as an instance for VERTEX COVER, returns the complement graph $(\overline{G}, n - k)$ as an equivalent instance of the INDEPENDENT SET problem in $\mathcal{O}(n^2)$ time. So, this is a polynomial time reduction algorithm. However, it is not a parameterised reduction as $k' = n - k$ cannot be bounded by any computable function $g(k)$ as n goes to infinity.

2.2.2 The W-hierarchy

We shall now try to characterise problems which are fixed-parameter intractable, thereby briefly introducing the notions of W-complexity classes.

The basis of fixed-parameter intractability entirely sits on the fact that no FPT algorithm is presently known for solving the CLIQUE problem. Rather, every result that shall be

mentioned in this section, holds true only under the hypothesis that *an FPT algorithm does not exist for the CLIQUE problem*. Under this hypothesis, and by the contrapositive of Lemma 2, if the CLIQUE problem can be FPT-reduced to any parameterised problem P , then P would be fixed-parameter intractable as well.

However, one gets to see a novel phenomenon in the case of these fixed-parameter intractable problems - a *hierarchy of complexity classes*. In the theory of classical complexity, all NP-complete problems are polynomially equivalent to each another. This does not happen in the case of parameterised reductions. For example, there is an FPT-reduction algorithm from the MULTICOLOURED INDEPENDENT SET problem to the DOMINATING SET problem (see, e.g. Theorem 13.9 [11]), but, presently no FPT reduction is known to exist in the opposite direction. Thus, the DOMINATING SET problem cannot be equivalent to the MULTICOLOURED INDEPENDENT SET problem, but it is strictly FPT-harder. This indicates the presence of a computationally harder subclass within the class of fixed parameter intractable problems, that captures this apparent hierarchy. These complexity sub-classes were defined and introduced by Downey and Fellows, [12] thereby giving rise to the *W-hierarchy of complexity classes*. Most of the intractable problems fall in the first three sub-classes of this hierarchy. However, as mentioned later by Downey and Fellows themselves, these complexity classes have limited practical use, and thus we would not dwell much into the technicalities of the W-hierarchy. We provide a brief description of the same, below:

To define the sub-classes within the W-hierarchy, we begin by defining a *Boolean circuit*. A Boolean circuit is a directed acyclic graph that tries to imitate a real life electric circuit made of logic-gates. In this sense, the nodes of a Boolean circuit are partitioned as follows:

- Input Nodes: Nodes with indegree 0
- NOT Nodes: Nodes with indegree 1
- AND or OR Nodes: Nodes with indegree atleast 2
- Output Node: Unique Node with outdegree 0

Similar to an electric circuit, assigning 0 or 1 values to the input nodes of such a Boolean circuit can uniquely determine the binary value of the output node. We say that this input assignment *satisfies* the circuit, if the resulting value of the output node equals 1. The sum

of all values of the input nodes is called the *weight of the input assignment*. The WEIGHTED CIRCUIT SATISFIABILITY (WCS) problem, takes as input a Boolean circuit C , parameter k and asks whether C has a *satisfying assignment of size at most k* . The CLIQUE problem can be reduced to the WCS problem, making the latter problem fixed-parameter intractable.

For any Boolean circuit, we can define the *depth d* of the circuit as the maximum length of a path from an input node to the output node. Further, we say a node is *large* if it has indegree of at least three. The *weft w* of a Boolean circuit is the maximum number of large nodes encountered on any path from an input node to the output node.

Now, let $C_{w,d}$ be the set of Boolean circuits with depth at most d and weft at most w , then, for every natural number n , we define the n^{th} W-hierarchy class as follows:

Definition 2.2.2. *For $n \in \mathbb{N}$, where $n \geq 1$, a parameterized problem P is said to lie in the sub-class $W[n]$ if there is a parameterized reduction from $WCS[C_{n,d}]$ for some $d \geq 1$.*

Here, $WCS[C_{n,d}]$ refers to the *WCS* problem restricted to the class $C_{n,d}$. The above sub-classes form a sequence of complexity classes for fixed parameter intractable problems and are collectively called the W-hierarchy. Further, it can be proven via FPT-reductions that:

$$FPT = W[0] \subset W[1] \subset W[2] \subset W[3] \subset W[4] \subset \dots$$

Finally, a parameterised problem P is said to be *complete* for a class $W[n]$ if $P \in W[n]$ and there is a parameterized reduction from every problem in $W[n]$ to P . The problems CLIQUE and INDEPENDENT SET are both $W[1]$ -complete, while the DOMINATING SET problem and its variations are $W[2]$ -complete. We refer to [11, 12] for further details on parameterized complexity.

Chapter 3

Defensive and Offensive Alliances

3.1 Introduction

In this chapter, we design FPT algorithms for DEFENSIVE ALLIANCE and OFFENSIVE ALLIANCE when parameterized by neighbourhood diversity of the input graph. We also design an FPT algorithm for OFFENSIVE ALLIANCE when parameterized by domino treewidth of the input graph. Finally we prove that DEFENSIVE ALLIANCE and OFFENSIVE ALLIANCE are polynomial time solvable for graphs with bounded treewidth. See also [25].

3.2 FPT algorithm parameterized by neighbourhood diversity

In this section, we present FPT algorithms for the DEFENSIVE ALLIANCE and OFFENSIVE ALLIANCE problems parameterized by neighbourhood diversity. It is known that the problems are fixed parameter tractable when parameterized by vertex cover number [42]. We prove that the problems remain fixed parameter tractable when parameterized by neighbourhood diversity. See Section 1.7 for the definitions of neighbourhood diversity and type graph.

3.2.1 Defensive Alliance

In this subsection, we prove the following theorem:

Theorem 3.2.1. DEFENSIVE ALLIANCE is *fixed-parameter-tractable* when parameterized by the neighbourhood diversity.

Given a graph $G = (V, E)$ with neighbourhood diversity $nd(G) \leq k$, we first find a partition of the vertices into at most k type classes C_1, \dots, C_k . Let $\mathcal{C} = \{C_1, \dots, C_k\}$ and let n_i denote the number of vertices in C_i . The case where some C_i are singletons can be considered as cliques or independent sets. For simplicity, we consider singleton type classes as independent sets. Let H be the type graph of G .

ILP formulation: Our goal here is to find a smallest defensive alliance S of G . For each C_i , we associate a variable x_i that indicates $|S \cap C_i| = x_i$. As the vertices in C_i have the same neighbourhood, the variables x_i determine S uniquely, up to isomorphism. We define

$$\mathcal{C}^+ = \{C_i \in \mathcal{C} \mid C_i \cap S \neq \emptyset\}$$

and

$$\mathcal{C}^- = \{C_i \in \mathcal{C} \mid C_i \cap S = \emptyset\}.$$

Note that S contains $x_i > 0$ vertices from class C_i if $C_i \in \mathcal{C}^+$ and contains no vertices from class C_i if $C_i \notin \mathcal{C}^+$. We next guess if a type class C_i belongs to \mathcal{C}^+ or \mathcal{C}^- . There are at most 2^k guesses as each clique type class C_i has two options: either it is in \mathcal{C}^+ or in \mathcal{C}^- . We reduce the problem of finding a smallest defensive alliance to at most 2^k integer linear programming problems with at most k variables. Since integer linear programming is fixed-parameter tractable when parameterized by the number of variables [47], we conclude that our problem is FPT when parameterized by the neighbourhood diversity k . We consider the following cases based on whether C_i is a clique type class or an independent type class.

Case 1: Let $C_i \in \mathcal{C}^+$ be an independent type class. Let $v \in V(C_i) \cap S$. Then the number

of neighbours of v in S is

$$d_S(v) = \sum_{C_j \in N_H(C_i) \cap \mathcal{C}^+} x_j \quad (3.1)$$

where $N_H(C_i) = \{C_j : (C_i, C_j) \in E(H)\}$. Thus, including itself, v has $1 + \sum_{C_i \in N_H(C_j) \cap \mathcal{C}^+} x_i$ defenders in G . Note that if $C_i \in \mathcal{C}^+$, then only x_i vertices of C_i are in S and the remaining $n_i - x_i$ vertices of C_i are outside S . The number of neighbours of v outside S is

$$d_{S^c}(v) = \sum_{C_j \in N_H(C_i) \cap \mathcal{C}^+} (n_j - x_j) + \sum_{C_j \in N_H(C_i) \cap \mathcal{C}^-} n_j \quad (3.2)$$

Therefore, a vertex v from an independent type class $C_i \in \mathcal{C}^+$ is protected if and only if

$$1 + \sum_{C_j \in N_H(C_i) \cap \mathcal{C}^+} x_j \geq \sum_{C_j \in N_H(C_i) \cap \mathcal{C}^+} (n_j - x_j) + \sum_{C_j \in N_H(C_i) \cap \mathcal{C}^-} n_j. \quad (3.3)$$

Equation 3.3 implies a vertex v from an independent type class $C_i \in \mathcal{C}^+$ is protected if and only if

$$1 + \sum_{C_j \in N_H(C_i) \cap \mathcal{C}^+} 2x_j \geq \sum_{C_j \in N_H(C_i)} n_j.$$

Case 2: Let $C_i \in \mathcal{C}^+$ be a clique type class. Let $v \in V(C_i) \cap S$. Then the number of neighbours of v in S is

$$d_S(v) = (x_i - 1) + \sum_{C_j \in N_H(C_i) \cap \mathcal{C}^+} x_j \quad (3.4)$$

This is to ensure that when $v \in V(C_i)$ is picked in the solution it contributes $x_i - 1$ to the value of $d_S(v)$ as v itself can't be accounted as its own neighbour. Therefore, including itself, v has $x_i + \sum_{C_j \in N_H(C_i) \cap \mathcal{C}^+} x_j$ neighbours in S . This can be written as

$$d_S(v) + 1 = \sum_{C_j \in N_H[C_i] \cap \mathcal{C}^+} x_j \quad (3.5)$$

where $N_H[C_i]$ denotes the closed neighbourhood of C_i in H . The number of neighbours of v

outside S is

$$d_{S^c}(v) = \sum_{C_j \in N_H[C_i] \cap \mathcal{C}^+} (n_j - x_j) + \sum_{C_j \in N_H[C_i] \cap \mathcal{C}^-} n_j \quad (3.6)$$

Thus a vertex v from clique type class $C_i \in \mathcal{C}$ is protected if and only if $d_S(v) + 1 \geq d_{S^c}(v)$, that is,

$$\sum_{C_j \in N_H[C_i] \cap \mathcal{C}^+} x_j \geq \sum_{C_j \in N_H[C_i] \cap \mathcal{C}^+} (n_j - x_j) + \sum_{C_j \in N_H[C_i] \cap \mathcal{C}^-} n_j. \quad (3.7)$$

Equation 3.7 implies, a vertex v from clique type class $C_i \in \mathcal{C}$ is protected if and only if

$$\sum_{C_j \in N_H[C_i] \cap \mathcal{C}^+} 2x_j \geq \sum_{C_j \in N_H[C_i]} n_j. \quad (3.8)$$

In the following, we present ILP formulation of defensive alliance problem, where \mathcal{C}^+ is given:

$$\begin{aligned} & \text{minimize } \sum_{C_i \in \mathcal{C}^+} x_i \\ & \text{subject to} \\ & 1 + \sum_{C_j \in N_H(C_i) \cap \mathcal{C}^+} 2x_j \geq \sum_{C_j \in N_H(C_i)} n_j \text{ for each independent type class } C_i \in \mathcal{C}^+ \\ & \sum_{C_j \in N_H[C_i] \cap \mathcal{C}^+} 2x_j \geq \sum_{C_j \in N_H[C_i]} n_j \text{ for each clique type class } C_i \in \mathcal{C}^+ \\ & x_i \in \{1, 2, \dots, n_i\} \text{ for all } i : C_i \in \mathcal{C}^+. \end{aligned}$$

Solving the ILP We have at most k variables in the ILP formulation of DEFENSIVE ALLIANCE. The value of objective function is bounded by n and the value of any variable in the integer linear programming is also bounded by n . The constraints can be represented using $O(k^2 \log n)$ bits. Lemma 1 of Section 2.1.3 implies that we can solve the problem with the guess \mathcal{C}^+ in FPT time. There are at most 2^k choices for \mathcal{C}^+ , and the ILP formula for a guess can be solved in FPT time. Thus Theorem 3.2.1 holds.

Example 4. Consider the graph shown in Figure 3.1. The type classes are $C_1 = \{a, b, c, d, e\}$, $C_2 = \{f, g, h\}$ and $C_3 = \{i, j, k, l\}$. We consider all possible guesses:

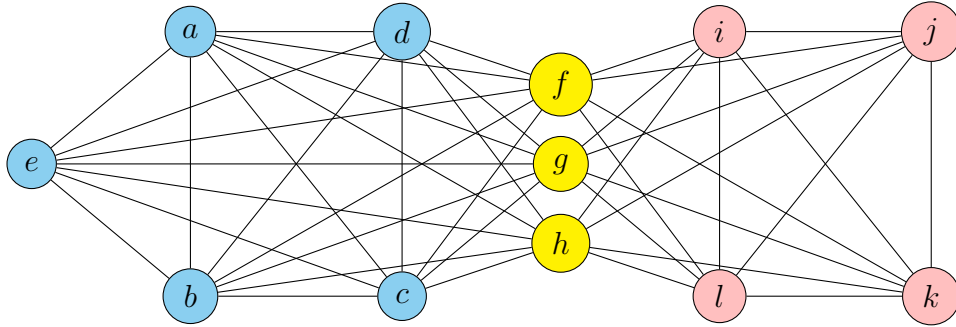


Figure 3.1: The graph in Example 4.

Guess 1: $\mathcal{C}^+ = \{C_1\}$. Then we have the following ILP:

$$\begin{aligned} \min \quad & x_1 \\ \text{subject to} \quad & x_1 \geq 4 \\ & x_1 \in \{1, 2, 3, 4, 5\} \end{aligned}$$

It is easy to see that $x_1 = 4$ is the optimal solution for the ILP. Thus $x_1 = 4, x_2 = 0, x_3 = 0$ represent a valid defensive alliance $\{a, b, c, d\}$ of size 4 for the graph.

Guess 2: $\mathcal{C}^+ = \{C_2\}$. In this case we do not get any valid constraints for x_2 .

Guess 3: $\mathcal{C}^+ = \{C_3\}$. Then we have the following ILP:

$$\begin{aligned} \min \quad & x_3 \\ \text{subject to} \quad & x_3 \geq 4 \\ & x_3 \in \{1, 2, 3, 4\} \end{aligned}$$

It is easy to see that $x_3 = 4$ is the optimal solution of the ILP. Thus $x_1 = 0, x_2 = 0, x_3 = 4$ represent a valid defensive alliance $\{i, j, k, l\}$ of size 4 for the graph.

Guess 4: $\mathcal{C}^+ = \{C_1, C_2\}$. Then we have the following ILP:

$$\begin{aligned} \min \quad & x_1 + x_2 \\ \text{subject to} \quad & x_1 \geq 4 \\ & x_1 + x_2 \geq 4 \\ & x_1 \in \{1, 2, 3, 4, 5\}; x_2 \in \{1, 2, 3\} \end{aligned}$$

It is easy to see that $x_1 = 4, x_2 = 1$ is the optimal solution of the ILP. Thus $x_1 = 4, x_2 = 1, x_3 = 0$ represent a valid defensive alliance $\{a, b, c, d, f\}$ of size 5 for the graph.

Guess 5: $\mathcal{C}^+ = \{C_1, C_3\}$. Then we have the following ILP:

$$\begin{aligned} \min \quad & x_1 + x_3 \\ \text{subject to} \quad & x_1 \geq 4 \\ & x_3 \geq 4 \\ & x_1 \in \{1, 2, 3, 4, 5\}; x_3 \in \{1, 2, 3, 4\} \end{aligned}$$

It is easy to see that $x_1 = 4, x_3 = 4$ is an optimal solution of the ILP. Thus $x_1 = 4, x_2 = 0, x_3 = 4$ represent a valid defensive alliance $\{a, b, c, d, i, j, k, l\}$ of size 8 for the graph.

Guess 6: $\mathcal{C}^+ = \{C_2, C_3\}$. Then we have the following ILP:

$$\begin{aligned} \min \quad & x_2 + x_3 \\ \text{subject to} \quad & x_3 \geq 4 \\ & x_2 + x_3 \geq 4 \\ & x_2 \in \{1, 2, 3\}; x_3 \in \{1, 2, 3, 4\} \end{aligned}$$

It is easy to see that $x_2 = 1, x_3 = 4$ is the optimal solution of the ILP. Thus $x_1 = 0, x_2 = 1, x_3 = 4$ represent a valid defensive alliance $\{f, i, j, k, l\}$ of size 5 for the graph.

Guess 7: $\mathcal{C}^+ = \{C_1, C_2, C_3\}$. Then we have the following ILP:

$$\begin{aligned}
& \min && x_1 + x_2 + x_3 \\
& \text{subject to} && x_1 + x_3 \geq 4 \\
& && x_1 + x_2 \geq 4 \\
& && x_2 + x_3 \geq 4 \\
& && x_1 \in \{1, 2, 3, 4, 5\}; x_2 \in \{1, 2, 3\}; x_3 \in \{1, 2, 3, 4\}
\end{aligned}$$

It is easy to see that $x_1 = 2, x_2 = 2, x_3 = 2$ is an optimal solution of the ILP. Thus $x_1 = 2, x_2 = 2, x_3 = 2$ represent a valid defensive alliance $\{a, b, f, g, i, j\}$ of size 6 for the graph.

Therefore the size of a smallest defensive alliance for the graph is 4.

3.2.2 Offensive Alliance

We also obtain the following result:

Theorem 3.2.2. OFFENSIVE ALLIANCE is fixed-parameter-tractable when parameterized by the neighbourhood diversity.

Proof. We give an ILP formulation of offensive alliance problem for a given \mathcal{C}^+ . Recall that a non-empty set $S \subseteq V$ is an offensive alliance in G if for each $v \in N(S)$, $d_S(v) \geq d_{S^c}(v) + 1$. The proof of this Theorem is similar to that of Theorem 3.2.1. We consider the following cases:

Case 1: Let $C_i \in N_H(\mathcal{C}^+)$ be an independent type class and $v \in C_i$. Then

$$d_S(v) = \sum_{C_j \in N_H(C_i) \cap \mathcal{C}^+} x_j \tag{3.9}$$

and

$$d_{S^c}(v) = \sum_{C_j \in N_H(C_i) \cap \mathcal{C}^+} (n_j - x_j) + \sum_{C_j \in N_H(C_i) \cap \mathcal{C}^-} n_j \tag{3.10}$$

Therefore, a vertex v from an independent type class $C_i \in N(\mathcal{C}^+)$ is protected if and only if $d_S(v) \geq d_{S^c}(v) + 1$, that is,

$$\sum_{C_j \in N_H(C_i) \cap \mathcal{C}^+} x_j \geq 1 + \sum_{C_j \in N_H(C_i) \cap \mathcal{C}^+} (n_j - x_j) + \sum_{C_j \in N_H(C_i) \cap \mathcal{C}^-} n_j. \quad (3.11)$$

Equation 3.11 implies a vertex v from an independent type class $C_i \in N(\mathcal{C}^+)$ is protected if and only if

$$\sum_{C_j \in N_H(C_i) \cap \mathcal{C}^+} 2x_j \geq 1 + \sum_{C_j \in N_H(C_i)} n_j. \quad (3.12)$$

Case 2: Let $C_i \in N_H(\mathcal{C}^+)$ be a clique type class and $v \in C_i$. Then

$$d_S(v) = \sum_{C_j \in N_H(C_i) \cap \mathcal{C}^+} x_j \quad (3.13)$$

and

$$d_{S^c}(v) = (n_i - 1) + \sum_{C_j \in N_H(C_i) \cap \mathcal{C}^+} (n_j - x_j) + \sum_{C_j \in N_H(C_i) \cap \mathcal{C}^-} n_j \quad (3.14)$$

Therefore, a vertex v from a clique type class $C_i \in N(\mathcal{C}^+)$ is protected if and only if $d_S(v) \geq d_{S^c}(v) + 1$, that is,

$$\sum_{C_j \in N_H(C_i) \cap \mathcal{C}^+} x_j \geq 1 + (n_i - 1) + \sum_{C_j \in N_H(C_i) \cap \mathcal{C}^+} (n_j - x_j) + \sum_{C_j \in N_H(C_i) \cap \mathcal{C}^-} n_j. \quad (3.15)$$

Equation 3.15 implies a vertex v from a clique type class $C_i \in N(\mathcal{C}^+)$ is protected if and only if

$$\sum_{C_j \in N_H(C_i) \cap \mathcal{C}^+} 2x_j \geq \sum_{C_j \in N_H[C_i]} n_j. \quad (3.16)$$

Case 3: Let $C_i \in \mathcal{C}^+$ be an independent type class and $v \in C_i \cap S^c$. It may be verified that v is protected if and only if it satisfies the condition in Equation 3.12.

Case 4: Let $C_i \in \mathcal{C}^+$ be a clique type class and $v \in C_i \cap S^c$. Then we have

$$d_S(v) = \sum_{C_j \in N_H[C_i] \cap \mathcal{C}^+} x_j \quad (3.17)$$

and

$$d_{S^c}(v) = (n_i - x_i) - 1 + \sum_{C_j \in N_H(C_i) \cap \mathcal{C}^+} (n_j - x_j) + \sum_{C_j \in N_H(C_i) \cap \mathcal{C}^-} n_j \quad (3.18)$$

Therefore, a vertex v from a clique type class $C_i \in \mathcal{C}^+$ is protected if and only if $d_S(v) \geq d_{S^c}(v) + 1$, that is,

$$\sum_{C_j \in N_H[C_i] \cap \mathcal{C}^+} x_j \geq (n_i - x_i) + \sum_{C_j \in N_H(C_i) \cap \mathcal{C}^+} (n_j - x_j) + \sum_{C_j \in N_H(C_i) \cap \mathcal{C}^-} n_j. \quad (3.19)$$

Equation 3.19 implies a vertex v from a clique type class $C_i \in \mathcal{C}^+$ is protected if and only if

$$\sum_{C_j \in N_H[C_i] \cap \mathcal{C}^+} 2x_j \geq \sum_{C_j \in N_H[C_i]} n_j. \quad (3.20)$$

We now present ILP formulation of offensive alliance problem for a given \mathcal{C}^+ .

$$\begin{aligned} & \text{minimize } \sum_{C_i \in \mathcal{C}^+} x_i \\ & \text{subject to} \\ & \sum_{C_j \in N_H(C_i) \cap \mathcal{C}^+} 2x_j \geq 1 + \sum_{C_j \in N_H(C_i)} n_j \text{ for each independent type class } C_i \in N_H(\mathcal{C}^+) \cup \mathcal{C}^+ \\ & \sum_{C_j \in N_H(C_i) \cap \mathcal{C}^+} 2x_j \geq \sum_{C_j \in N_H[C_i]} n_j \text{ for each clique type class } C_i \in N_H(\mathcal{C}^+) \\ & \sum_{C_j \in N_H[C_i] \cap \mathcal{C}^+} 2x_j \geq \sum_{C_j \in N_H[C_i]} n_j \text{ for each clique type class } C_i \in \mathcal{C}^+ \\ & x_i \in \{1, 2, \dots, n_i\} \text{ for all } i : C_i \in \mathcal{C}^+. \end{aligned}$$

In the formulation for OFFENSIVE ALLIANCE problem, we have at most k variables. The

value of objective function is bounded by n and the value of any variable in the integer linear programming is also bounded by n . The constraints can be represented using $O(k^2 \log n)$ bits. Proposition ?? implies that we can solve the problem with the guess \mathcal{C}^+ in FPT time. There are at most 2^k guesses, and the ILP formula for a guess can be solved in FPT time. Thus Theorem 3.2.2 holds. \square

Example 5. Consider the graph shown in Figure 3.1. The type classes are $C_1 = \{a, b, c, d, e\}$, $C_2 = \{f, g, h\}$ and $C_3 = \{i, j, k, l\}$. We consider all possible guesses:

Guess 1: $\mathcal{C}^+ = \{C_1\}$. Then we have the following ILP:

$$\begin{aligned} \min \quad & x_1 \\ \text{subject to} \quad & x_1 \geq 5 \\ & x_1 \in \{1, 2, 3, 4, 5\} \end{aligned}$$

It is easy to see that $x_1 = 5$ is the optimal solution for the ILP. Thus $x_1 = 5, x_2 = 0, x_3 = 0$ represent a valid offensive alliance $\{a, b, c, d, e\}$ of size 5 for the graph.

Guess 2: $\mathcal{C}^+ = \{C_2\}$. Then we have the following ILP:

$$\begin{aligned} \min \quad & x_2 \\ \text{subject to} \quad & x_2 \geq 4 \\ & x_2 \in \{1, 2, 3\} \end{aligned}$$

In this case we do not get any valid constraints for x_2 ; so no solution.

Guess 3: $\mathcal{C}^+ = \{C_3\}$. Then we have the following ILP:

$$\begin{aligned} \min \quad & x_3 \\ \text{subject to} \quad & x_3 \geq 5 \\ & x_3 \in \{1, 2, 3, 4\} \end{aligned}$$

In this case we do not get any valid constraints for x_3 ; so no solution.

Guess 4: $\mathcal{C}^+ = \{C_1, C_2\}$. Then we have the following ILP:

$$\begin{aligned} \min \quad & x_1 + x_2 \\ \text{subject to} \quad & x_1 \geq 5 \\ & x_2 \geq 4 \\ & x_1 \in \{1, 2, 3, 4, 5\}; x_2 \in \{1, 2, 3\} \end{aligned}$$

In this case we do not get any valid constraints for x_2 ; no solution.

Guess 5: $\mathcal{C}^+ = \{C_1, C_3\}$. Then we have the following ILP:

$$\begin{aligned} \min \quad & x_1 + x_3 \\ \text{subject to} \quad & x_1 \geq 4 \\ & x_3 \geq 4 \\ & x_1 \in \{1, 2, 3, 4, 5\}; x_3 \in \{1, 2, 3, 4\} \end{aligned}$$

It is easy to see that $x_1 = 4, x_3 = 4$ is an optimal solution of the ILP. Thus $x_1 = 4, x_2 = 0, x_3 = 4$ represent a valid offensive alliance $\{a, b, c, d, i, j, k, l\}$ of size 8 for the graph.

Guess 6: $\mathcal{C}^+ = \{C_2, C_3\}$. Then we have the following ILP:

$$\begin{aligned} \min \quad & x_2 + x_3 \\ \text{subject to} \quad & x_2 \geq 4 \\ & x_3 \geq 4 \\ & x_2 \in \{1, 2, 3\}; x_3 \in \{1, 2, 3, 4\} \end{aligned}$$

In this case we do not get any valid constraints for x_2 ; no solution.

Guess 7: $\mathcal{C}^+ = \{C_1, C_2, C_3\}$. Then we have the following ILP:

$$\begin{aligned}
& \min && x_1 + x_2 + x_3 \\
& \text{subject to} && x_1 + x_3 \geq 5 \\
& && x_1 + x_2 \geq 4 \\
& && x_2 + x_3 \geq 4 \\
& && x_1 \in \{1, 2, 3, 4, 5\}; x_2 \in \{1, 2, 3\}; x_3 \in \{1, 2, 3, 4\}
\end{aligned}$$

It is easy to see that $x_1 = 2, x_2 = 2, x_3 = 3$ is an optimal solution of the ILP. Thus $x_1 = 2, x_2 = 2, x_3 = 3$ represent a valid offensive alliance $\{a, b, f, g, i, j, k\}$ of size 7 for the graph.

Therefore the size of a smallest defensive alliance for the graph is 5.

3.3 FPT algorithm parameterized by domino treewidth

It is known that the DEFENSIVE ALLIANCE and OFFENSIVE ALLIANCE problems are W[1]-hard when parameterized by treewidth of the input graph [6, 22], which rule out FPT algorithms under common assumptions. This puts these two problems among the few problems that are FPT when parameterized by solution size but not FPT when parameterized by treewidth (unless FPT = W[1]). Thus we look at domino treewidth. Enciso [13] proved that finding defensive and global defensive alliances are fixed parameter tractable when parameterized by domino treewidth. In this section, we show that when parameterized by domino treewidth d , the problem of finding smallest offensive alliance is fixed parameter tractable. See Section 1.7 for definitions of tree decomposition and treewidth.

It is important to note that a graph may have several different tree decomposition. Similarly, the same tree decomposition can be valid for several different graphs. Every graph has a trivial tree decomposition for which T has only one vertex including all of V . However, this is not effective for the purpose of solving problems.

Definition 3.3.1. [8] A tree decomposition $(T, \{X_t\}_{t \in V(T)})$ is a *domino tree decomposition* if for $i, j \in V(T)$ where $i \neq j$ and $(i, j) \notin E(T)$, then $X_i \cap X_j = \emptyset$. In other words, in a domino tree decomposition, every vertex of G appears in at most two bags in T .

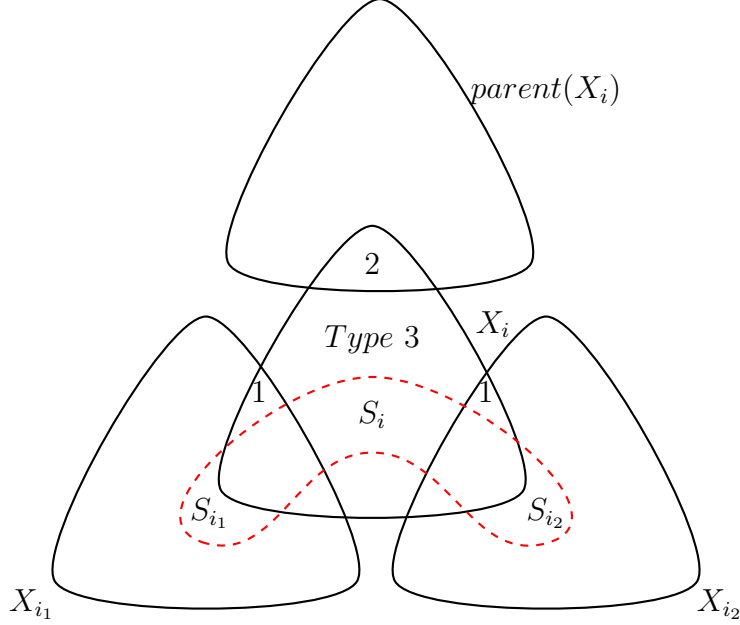


Figure 3.2: Compatibility of S_i with S_{i_1} and S_{i_2} . Set S_i is the region bounded by the dotted line in X_i ; S_{i_j} is the region bounded by the dotted line in X_{i_j} for $j = 1, 2$.

The *width* of a tree decomposition is defined as $width(T) = \max_{t \in V(T)} |X_t| - 1$ and the treewidth $tw(G)$ of a graph G is the minimum width among all possible tree decompositions of G . Similarly, domino treewidth $dtw(G)$ is defined for domino tree decomposition. Note that the number of nodes in a domino tree decomposition remains order n . Moreover, given a graph G with treewidth k and maximum degree Δ , the domino treewidth $dtw(G) \leq (9k + 7)\Delta(\Delta + 1)$, can be obtained in polynomial time [8].

Let $\tau = (T, \{X_t\}_{t \in V(T)})$ be a domino tree decomposition of the input n -vertex graph G that has width at most d . Suppose T is rooted at node r and $X_r = \emptyset$. For a node i of T , let V_i be the union of all bags present in the subtree of T rooted at i , including X_i . We denote by G_i the subgraph of G induced by V_i .

Let X_i be a non-leaf bag. Then a vertex $v \in X_i$ can be of three types. *Type 1*: v is also in one of the children of X_i ; *Type 2*: v is also in the parent of X_i ; *Type 3*: v is only in X_i .

For every bag i and every $S_i \subseteq X_i$, a *potential offensive alliance* (pOA) is a smallest set \widehat{S}_i

such that $S_i \subseteq \widehat{S}_i \subseteq V_i$, $\widehat{S}_i \cap X_i = S_i$, and \widehat{S}_i protects the vertices of $N_{V_i}(\widehat{S}_i) - \text{parent}(X_i)$. We use $c[i, S_i]$ to denote the size of \widehat{S}_i . If no such set \widehat{S}_i exists, then we put $c[i, S_i] = |\widehat{S}_i| = \infty$. We now move on to presenting how the values of $c[., .]$ are computed. We compute the values of $c[., .]$ at each node i based on the values computed for the children of i . We give a recursive formula for the computation of $c[., .]$. The values of $c[., .]$ for leaf node corresponds to the base case of the recurrence; whereas the values of $c[., .]$ for a non-leaf node i depend on the values of $c[., .]$ for the children of i . We finally compute $c[r, \emptyset]$ by applying the formulas in a bottom-up manner on T . Note that $c[r, \emptyset]$ is the size of minimum offensive alliance in G ; this is due to the fact that $V_r = V(G)$ and $S_r = \emptyset$.

Leaf node: If i is a leaf node, then for every $S_i \subseteq X_i \neq \emptyset$, we define $c[i, S_i]$ as follows:

$$c[i, S_i] = \begin{cases} |S_i| & \text{if } S_i \text{ is a non empty subset of } X_i \text{ and protects} \\ & \text{all vertices in } N_{X_i}(S_i) - \text{parent}(X_i) \\ \infty & \text{otherwise.} \end{cases}$$

Non leaf node: Suppose i is a non-leaf node with two children i_1 and i_2 . We say that a set $S_i \subseteq X_i$ is compatible with $S_{i_1} \subseteq X_{i_1}, S_{i_2} \subseteq X_{i_2}$ if and only if

1. $X_i \cap X_{i_j} \cap S_i = X_i \cap X_{i_j} \cap S_{i_j}$ for $1 \leq j \leq 2$.
2. *Type 1* and *Type 3* vertices of $N_{X_i}(S_i \cup S_{i_1} \cup S_{i_2})$ are protected by $S_i \cup S_{i_1} \cup S_{i_2}$.

For $S_i \subseteq X_i$, if there does not exist any $S_{i_j} \subseteq X_{i_j}$ that is compatible with S_i for some j , then $c[i, S_i] = \infty$. Otherwise, $c[i, S_i] =$

$$|S_i| + \min \left\{ \sum_{j=1}^2 c[i_j, S_{i_j}] - |S_i \cap S_{i_j}| : S_{i_j} \subseteq X_{i_j}; S_{i_1}, S_{i_2} \text{ are compatible with } S_i \right\}.$$

Theorem 3.3.1. For every node i in T and every $S_i \subseteq X_i$, $c[i, S_i]$ is the size of the smallest potential offensive alliance \widehat{S}_i where $\widehat{S}_i \subseteq V_i$ and $\widehat{S}_i \cap X_i = S_i$. Further, the size of the minimum offensive alliance in G is $c[r, \emptyset]$, where T is rooted at node r .

Proof. We will prove the theorem by induction. When node i is a leaf node, for every $S_i \subseteq X_i$, clearly $c[i, S_i]$ is the size of a smallest potential defensive alliance as $\widehat{S}_i = S_i$. So the statement is true for leaf node i .

Let i be a non-leaf node of T and let i_1 and i_2 be two children of i . Let T_{i_1} and T_{i_2} be the subtrees rooted at i_1 and i_2 respectively. By induction hypothesis, for arbitrary $S_{i_j} \subseteq X_{i_j}$, $c[i_j, S_{i_j}]$ is the size of a smallest potential offensive alliance \widehat{S}_{i_j} , where $S_{i_j} = \widehat{S}_{i_j} \cap X_{i_j}$ for $i = 1, 2$. We need to show that for arbitrary $S_i \subseteq X_i$, $c[i, S_i]$ as computed by the above recurrence relation is the size of a smallest potential offensive alliance $\widehat{S}_i \subseteq V_i$, where $S_i = \widehat{S}_i \cap X_i$. If there does not exist $S_{i_1} \subseteq X_{i_1}$ and $S_{i_2} \subseteq X_{i_2}$ that are compatible with S_i , then $c[i, S_i] = \infty$. Otherwise, $c[i, S_i] =$

$|S_i| + \min \left\{ \sum_{j=1}^2 c[i_j, S_{i_j}] - |S_i \cap S_{i_j}| : S_{i_j} \subseteq X_{i_j}; S_{i_1}, S_{i_2} \text{ are compatible with } S_i \right\}$. By induction hypothesis for any subset $S_{i_j} \subseteq X_{i_j}$, $c[i_j, S_{i_j}]$ is the size of a minimum potential offensive alliance \widehat{S}_{i_j} , where $S_{i_j} = \widehat{S}_{i_j} \cap X_{i_j}$ and the relation considers all possible compatible sets.

To prove \widehat{S}_i is a potential offensive alliance, we must prove that \widehat{S}_i protects all the vertices $v \in N(\widehat{S}_i) - \text{parent}(X_i)$. Suppose $\widehat{S}_i = S_i \cup \widehat{S}_{i_1} \cup \widehat{S}_{i_2}$. Since S_{i_1} and S_{i_2} are compatible with S_i all the vertices of *Type 1* and *Type 3* in $N_{X_i}(S_i \cup S_{i_1} \cup S_{i_2})$ are protected by $S_i \cup S_{i_1} \cup S_{i_2}$. By definition \widehat{S}_{i_j} protects all vertices in $N(\widehat{S}_{i_j}) - \text{parent}(X_{i_j})$. Moreover, since S_{i_1} and S_{i_2} are compatible with S_i , the vertices in $X_{i_1} \cup X_{i_2}$ that were of *Type 2* are of *Type 1* in X_i and $S_i \cup S_{i_1} \cup S_{i_2}$ protects all *Type 1* vertices of $N_{V_i}(S_i \cup S_{i_1} \cup S_{i_2})$. Thus \widehat{S}_i protects all the vertices in $N_{V_i}(\widehat{S}_i) - \text{parent}(X_i)$. Now we prove that $c[i, S_i]$ is the size of a smallest potential offensive alliance \widehat{S}_i such that $S_i = \widehat{S}_i \cap X_i$. From induction hypothesis, $c[i_j, S_{i_j}]$ is the size of a smallest potential offensive alliance \widehat{S}_{i_j} such that $S_{i_j} = \widehat{S}_{i_j} \cap X_{i_j}$, for $j = 1, 2$. For computation of $c[i, S_i]$ all possible subsets $S_{i_1} \subseteq X_{i_1}$ and $S_{i_2} \subseteq X_{i_2}$ that are compatible with S_i are evaluated, thus $c[i, S_i]$ is the size of a smallest potential offensive alliance \widehat{S}_i such that $S_i = \widehat{S}_i \cap X_i$.

Now we prove that $c[r, \emptyset]$ is the size of a minimum offensive alliance in G . Let $r - 1$ be the child of r . Since $X_r = \emptyset$, we have from the recurrence relation $c[r, \emptyset] = \min \left\{ c[r - 1, S_{r-1}] : S_{r-1} \subseteq X_{r-1}; S_r = \emptyset \text{ and } S_{r-1} \text{ are compatible} \right\}$. Since there are no *Type 2* vertices in X_{r-1} , $N_{V_{r-1}}(\widehat{S}_{r-1})$ is protected. For computation of $c[r, \emptyset]$ all subsets $S_{r-1} \subseteq X_{r-1}$ that are compatible with \emptyset are evaluated, thus $c[r, \emptyset]$ is the size of a minimum offensive alliance. This completes the proof of the theorem. \square

Note that the definition of compatibility and the above recurrence relation can be easily extended if a non-leaf node i has more than two children. At a non-leaf node we compute 2^{d+1} many $c[.,.]$ values and the time need to compute each of these values is $O(4^{d+1})$, assuming

binary domino tree decomposition. As the number of nodes in domino tree decomposition is $O(n)$, the total running time of the algorithm is $O(8^d n)$.

3.4 Graphs of bounded treewidth

It is known that the DEFENSIVE ALLIANCE and OFFENSIVE ALLIANCE problems are W[1]-hard when parameterized by treewidth of the input graph [6, 22], which rule out FPT algorithms under common assumptions. This was surprising as DEFENSIVE ALLIANCE is a “subset problem” and FPT when parameterized by solution size, and “subset problems” that satisfy this property usually tend to be FPT for bounded treewidth as well. In this section we prove that DEFENSIVE ALLIANCE and OFFENSIVE ALLIANCE problems can be solved in polynomial time for graphs of bounded treewidth. In other words, this section presents XP-time algorithms for DEFENSIVE ALLIANCE and OFFENSIVE ALLIANCE problems parameterized by treewidth. We design dynamic programming based XP-algorithms for DEFENSIVE ALLIANCE and OFFENSIVE ALLIANCE when parameterized by the treewidth of input graph. See Section 1.7, for definitions of tree decomposition and nice tree decomposition.

3.4.1 Defensive Alliance

In this subsection, we prove the following theorem:

Theorem 3.4.1. *Given an n -vertex graph G and its nice tree decomposition T of width at most k , the size of a minimum defensive alliance of G can be computed in $O(2^k n^{2k+4})$ time.*

Let $(T, \{X_t\}_{t \in V(T)})$ be a nice tree decomposition rooted at node r of the input graph G . For a node t of T , let V_t be the union of all bags present in the subtree of T rooted at t , including X_t . We denote by G_t the subgraph of G induced by V_t . For each node t of T , we construct a table $dp_t(A, \mathbf{n}, a, \alpha) \in \{\text{true}, \text{false}\}$ where $A \subseteq X_t$, \mathbf{n} is a vector of length n , and its i th coordinate is positive only if $v_i \in A$; a and α are integers between 0 and n . We set $dp_t(A, \mathbf{n}, a, \alpha) = \text{true}$ if and only if there exists a set $A_t \subseteq V_t$ such that:

1. $A_t \cap X_t = A$

2. $a = |A_t|$

3. the i th coordinate of vector \mathbf{n} is

$$n(i) = \begin{cases} d_{A_t}(v_i) & \text{for } v_i \in A \\ 0 & \text{otherwise} \end{cases}$$

4. α is the number of vertices $v \in A_t$ that are protected, that is, $d_{A_t}(v) \geq \frac{d_G(v)-1}{2}$.

We compute all entries $dp_t(A, \mathbf{n}, a, \alpha)$ in a bottom-up manner. Since $tw(T) \leq k$, at most $2^k n^k (n+1)^2 = O(2^k n^{k+2})$ records are maintained at each node t . Thus, to prove Theorem 5.7.1, it suffices to show that each entry $dp_t(A, \mathbf{n}, a, \alpha)$ can be computed in $O(n^{k+2})$ time, assuming that the entries for the children of t are already computed.

Leaf node: For leaf node t we have that $X_t = \emptyset$. Thus $dp_t(A, \mathbf{n}, a, \alpha)$ is true if and only if $A = \emptyset$, $\mathbf{n} = \mathbf{0}$, $a = 0$ and $\alpha = 0$. These conditions can be checked in $O(1)$ time.

Introduce node: Suppose t is an introduction node with child t' such that $X_t = X_{t'} \cup \{v_i\}$ for some $v_i \notin X_{t'}$. Let A be any subset of X_t . We consider two cases:

Case (i): Let $v_i \notin A$. In this case $dp_t(A, \mathbf{n}, a, \alpha)$ is true if and only if $dp_{t'}(A, \mathbf{n}, a, \alpha)$ is true.

Case (ii): Let $v_i \in A$. Here $dp_t(A, \mathbf{n}, a, \alpha)$ is true if and only if there exist A' , \mathbf{n}' , a' , and α' such that $dp_{t'}(A', \mathbf{n}', a', \alpha') = \text{true}$, where

1. $A = A' \cup \{v_i\}$;
2. $n(j) = n'(j) + 1$, if $v_j \in N_A(v_i)$, $n(i) = d_A(v_i)$, and $n(j) = n'(j)$ if $v_j \in A \setminus N_A[v_i]$;
3. $a = a' + 1$;
4. $\alpha = \alpha' + \delta$; here δ is the cardinality of the set

$$\left\{ v_j \in A \mid n'(j) < \frac{d_G(v_j) - 1}{2}; n(j) \geq \frac{d_G(v_j) - 1}{2} \right\}.$$

That is, to compute α from α' we need to add the number δ of vertices which are not satisfied in $(A', \mathbf{n}', a', \alpha')$ but satisfied in $(A, \mathbf{n}, a, \alpha)$.

For introduce node t , $dp_t(A, \mathbf{n}, a, \alpha)$ can be computed in $O(1)$ time. This follows from the fact that there is only one candidate of such tuple $(A', \mathbf{n}', a', \alpha')$.

Forget node: Suppose t is a forget node with child t' such that $X_t = X_{t'} \setminus \{v_i\}$ for some $v_i \in X_{t'}$. Let A be any subset of X_t . Here $dp_t(A, \mathbf{n}, a, \alpha)$ is true if and only if either $dp_{t'}(A, \mathbf{n}, a, \alpha)$ is true (this corresponds to the case that A_t does not contain v_i) or $dp_{t'}(A', \mathbf{n}', a, \alpha) = \text{true}$ for some A', \mathbf{n}' with the following conditions:

1. $A = A' \setminus \{v_i\}$;
2. $n(j) = n'(j)$ for all $j \neq i$ and $n(i) = 0$;

(this corresponds to the case that A_t contains v_i). For forget node t , $dp_t(A, \mathbf{n}, a, \alpha)$ can be computed in $O(n)$ time. This follows from the fact that there are $O(n)$ candidates of such tuple $(A', \mathbf{n}', a, \alpha)$.

Join node: Suppose t is a join node with children t_1 and t_2 such that $X_t = X_{t_1} = X_{t_2}$. Let A be any subset of X_t . Then $dp_t(A, \mathbf{n}, a, \alpha)$ is true if and only if there exist $(A_1, \mathbf{n}_1, a_1, \alpha_1)$ and $(A_2, \mathbf{n}_2, a_2, \alpha_2)$ such that $dp_{t_1}(A_1, \mathbf{n}_1, a_1, \alpha_1) = \text{true}$ and $dp_{t_2}(A_2, \mathbf{n}_2, a_2, \alpha_2) = \text{true}$, where

1. $A = A_1 = A_2$;
2. $n(i) = n_1(i) + n_2(i) - d_A(v_i)$ for all $i \in A$, and $n(i) = 0$ if $i \notin A$;
3. $a = a_1 + a_2 - |A|$;
4. $\alpha = \alpha_1 + \alpha_2 - \gamma + \delta$; γ is the cardinality of the set

$$\left\{ v_j \in A \mid n_1(j) \geq \frac{d_G(v_i) - 1}{2}; n_2(j) \geq \frac{d_G(v_i) - 1}{2} \right\}$$

and δ is the cardinality of the set

$$\left\{ v_j \in A \mid n_1(j) < \frac{d_G(v_i) - 1}{2}; n_2(j) < \frac{d_G(v_i) - 1}{2}; n(j) \geq \frac{d_G(v_i) - 1}{2} \right\}.$$

To compute α from $\alpha_1 + \alpha_2$, we need to subtract the number of those v_j which are satisfied in both the branches and add the number of vertices v_j not satisfied in either of the branches t_1 and t_1 but satisfied in t .

For join node t , there are n^k possible pairs for $(\mathbf{n}_1, \mathbf{n}_2)$ as \mathbf{n}_2 is uniquely determined by \mathbf{n}_1 ; $n + 1$ possible pairs for (a_1, a_2) ; and $n + 1$ possible pairs for (α_1, α_2) . In total, there are $O(n^{k+2})$ candidates, and each of them can be checked in $O(1)$ time. Thus, for join node t , $dp_t(A, \mathbf{n}, a, \alpha)$ can be computed in $O(n^{k+2})$ time.

At the root node r , we look at all records such that $dp_r(\emptyset, \mathbf{n}, a, \alpha) = \text{true}$, $a, \alpha > 0$ and $a = \alpha$. The size of a minimum defensive alliance is the minimum a satisfying $dp_r(\emptyset, \mathbf{n}, a, a) = \text{true}$ and $a > 0$.

3.4.2 Offensive Alliance

We also obtain the following result:

Theorem 3.4.2. *Given an n -vertex graph G and its nice tree decomposition T of width at most k , the size of a minimum offensive alliance of G can be computed in $O(2^k n^{2k+6})$ time.*

Proof. Let $(T, \{X_t\}_{t \in V(T)})$ be a nice tree decomposition rooted at node r of the input graph G . For each node t of T , we construct a table $dp_t(A, \mathbf{m}, a, b, \beta) \in \{\text{true}, \text{false}\}$ where $A \subseteq X_t$; \mathbf{m} is a vector of length n , all of its coordinates are integers between 0 and $n - 1$, and its i th coordinate is positive only if $v_i \in N_{X_t}(A_t)$; a, b and β are integers between 0 and n . We set $dp_t(A, \mathbf{m}, a, b, \beta) = \text{true}$ if and only if there exists a set $A_t \subseteq V_t$ such that:

1. $A_t \cap X_t = A$
2. $a = |A_t|$
3. $b = |N_{G_t}(A_t)|$

4. the i th coordinate of vector \mathbf{m} is

$$m(i) = \begin{cases} d_{A_t}(v_i) & \text{for } v_i \in N_{X_t}(A_t) \\ 0 & \text{otherwise} \end{cases}$$

5. β is the number of vertices $v \in N_{G_t}(A_t)$ that are protected by A_t , that is, $d_{A_t}(v) \geq \frac{d_G(v)+1}{2}$.

We compute all entries $dp_t(A, \mathbf{m}, a, b, \beta)$ in a bottom-up manner. Since $tw(T) \leq k$, at most $2^k \times n^k \times (n+1)^3 = O(2^k n^{k+3})$ records are maintained at each node t . Thus, to prove Theorem 3.4.2, it suffices to show that each entry $dp_t(A, \mathbf{m}, a, b, \beta)$ can be computed in $O(n^{k+3})$ time, assuming that the entries for the children of t are already computed.

Leaf node: For leaf node t we have that $X_t = \emptyset$. Thus $dp_t(A, \mathbf{m}, a, b, \beta)$ is true if and only if $A = \emptyset$, $\mathbf{m} = \mathbf{0}$, $a = 0$, $b = 0$ and $\beta = 0$. These conditions can be checked in $O(1)$ time.

Introduce node: Suppose t is an introduction node with child t' such that $X_t = X_{t'} \cup \{v_i\}$ for some $v_i \notin X_{t'}$. Let A be any subset of X_t . We consider three cases:

Case (i): $v_i \notin A$ and v_i is not adjacent to any vertex in A . In this case $dp_t(A, \mathbf{m}, a, b, \beta)$ is true if and only if $dp_{t'}(A, \mathbf{m}, a, b, \beta)$ is true.

Case (ii): Suppose $v_i \notin A$ and v_i is adjacent to a vertex in A . In this case $dp_t(A, \mathbf{m}, a, b, \beta)$ is true if and only if there exist $A', \mathbf{m}', a', b', \beta'$ such that $dp_{t'}(A', \mathbf{m}', a', b', \beta')$ is true, where

1. $A = A'$;
2. $m(i) = d_A(v_i)$, $m(j) = m'(j)$ for all $j \neq i$;
3. $a = a'$;
4. $b = b' + 1$;

5.

$$\beta = \begin{cases} \beta' + 1 & \text{if } d_{A_t}(v_i) \geq \frac{d_G(v)+1}{2} \\ \beta' & \text{otherwise} \end{cases}$$

Case (iii): Suppose $v_i \in A$. In this case $dp_t(A, \mathbf{m}, a, b, \beta)$ is true if and only if there exist $A', \mathbf{m}', a', b', \beta'$ such that $dp_{t'}(A', \mathbf{m}', a', b', \beta')$ is true, where

1. $A = A' \cup \{v_i\}$;

2.

$$m(j) = \begin{cases} m'(j) + 1 & \text{if } v_j \in N_{X_t}(A_t) \text{ and } (v_i, v_j) \in E(G) \\ m'(j) & \text{if } v_j \in N_{X_t}(A_t) \text{ and } (v_i, v_j) \notin E(G) \\ 0 & \text{if } v_j \notin N_{X_t}(A_t) \end{cases}$$

3. $a = a' + 1$

4. $b = b' + d_{A^c \cap X_t}(v_i)$

5. $\beta = \beta' + \delta$; here δ is the cardinality of the set

$$\left\{ v_j \in N_{X_t}(A_t) \mid m'(j) < \frac{d_G(v_j) + 1}{2}; m(j) \geq \frac{d_G(v_j) + 1}{2} \right\}.$$

That is, to compute β from β' we need to add the number δ of those vertices not protected in $(A', \mathbf{m}', a', b', \beta')$ but protected in $(A, \mathbf{m}, a, b, \beta)$.

For introduce node t , $dp_t(A, \mathbf{m}, a, b, \beta)$ can be computed in $O(1)$ time. This follows from the fact that there is only one candidate of such tuple $(A', \mathbf{m}', a', b', \beta')$.

Forget node: Suppose t is a forget node with child t' such that $X_t = X_{t'} \setminus \{v_i\}$ for some $v_i \in X_{t'}$. Let A be any subset of X_t . Here $dp_t(A, \mathbf{m}, a, b, \beta)$ is true if and only if either

1. there exists \mathbf{m}' such that $dp_{t'}(A, \mathbf{m}', a, b, \beta)$ is true, where $m(j) = m'(j)$ for all $j \neq i$; $m(i) = 0$; or

2. there exist A' such that $dp_{t'}(A', \mathbf{m}, a, b, \beta)$ is true, where $A = A' \setminus \{v_i\}$;

For forget node t , $dp_t(A, \mathbf{m}, a, b, \beta)$ can be computed in $O(n)$ time. This follows from the fact that there are $O(n)$ candidates of such tuple $(A, \mathbf{m}', a, b, \beta)$.

Join node: Suppose t is a join node with children t_1 and t_2 such that $X_t = X_{t_1} = X_{t_2}$. Let A be any subset of X_t . Then $dp_t(A, \mathbf{m}, a, b, \beta)$ is true if and only if there exist $A_1, \mathbf{m}_1, a_1, b_1, \beta_1, A_2, \mathbf{m}_2, a_2, b_2, \beta_2$ such that $dp_{t_1}(A_1, \mathbf{m}_1, a_1, b_1, \beta_1) = \text{true}$ and $dp_{t_2}(A_2, \mathbf{m}_2, a_2, b_2, \beta_2) = \text{true}$, where

1. $A = A_1 = A_2$;
2. $m(i) = m_1(i) + m_2(i) - d_A(v_i)$ if $\max\{m_1(i), m_2(i)\} > 0$, $m(i) = 0$, otherwise;
3. $a = a_1 + a_2 - |A|$;
4. $b = b_1 + b_2 - |N_{X_t}(A)|$;
5. $\beta = \beta_1 + \beta_2 - \gamma + \delta$; γ is the cardinality of the set

$$\left\{ v_j \in N_{X_t}(A) \mid m_1(j) \geq \frac{d_G(v_i) + 1}{2}; m_2(j) \geq \frac{d_G(v_i) + 1}{2} \right\}$$

and δ is the cardinality of the set

$$\left\{ v_j \in N_{X_t}(A) \mid m_1(j) < \frac{d_G(v_i) + 1}{2}; m_2(j) < \frac{d_G(v_i) + 1}{2}; m(j) \geq \frac{d_G(v_i) + 1}{2} \right\}.$$

To compute β from $\beta_1 + \beta_2$, we need to subtract the number of those v_j which are satisfied in both the branches and add the number of vertices v_j not satisfied in either of the branches but satisfied in t .

For join node t , there are n^k possible pairs for $(\mathbf{m}_1, \mathbf{m}_2)$ as \mathbf{m}_2 is uniquely determined by \mathbf{m}_1 ; $n + 1$ possible pairs for (a_1, a_2) , (b_1, b_2) and (β_1, β_2) each. In total, there are $O(n^{k+3})$ candidates, and each of them can be checked in $O(1)$ time. Thus, for join node t , $dp_t(A, \mathbf{m}, a, b, \beta)$ can be computed in $O(n^{k+3})$ time.

At the root node r , we look at all records such that $dp_r(\emptyset, \mathbf{m}, a, b, \beta) = \text{true}$, $a > 0$ and $b = \beta$. The size of a minimum offensive alliance is the minimum a satisfying $dp_r(\emptyset, \mathbf{m}, a, b, \beta) = \text{true}$ and $a > 0$. □

Chapter 4

Parameterized Intractability of Defensive Alliance Problem

4.1 Introduction

Bliem and Woltran [6] proved that defensive alliance problem is $W[1]$ -hard when parameterized by the treewidth of the input graph. However, for the parameter pathwidth, the question of whether the problem is FPT has remained open. In this chapter, we give a negative answer by showing that the problem is $W[1]$ -hard when parameterized by pathwidth, which rules out FPT algorithms under common assumptions. This chapter describes a parameterized reduction from the MULTIDIMENSIONAL SUBSET SUM (MSS) problem. On the way towards this result, we provide hardness results for two variants of the DEFENSIVE ALLIANCE problem, called $\text{DEFENSIVE ALLIANCE}^F$ and $\text{DEFENSIVE ALLIANCE}^{FN}$, which we believe are interesting in their own right. We prove that the $\text{DEFENSIVE ALLIANCE}^{FN}$ problem is $W[1]$ -hard when parameterized by the pathwidth of the input graph. We also prove that the $\text{DEFENSIVE ALLIANCE}^F$ problem is $W[1]$ -hard when parameterized by the pathwidth of the graph.

4.2 W[1]-hardness parameterized by pathwidth

In this section we prove the following theorems:

Theorem 4.2.1. The DEFENSIVE ALLIANCE problem is W[1]-hard when parameterized by the pathwidth of the input graph.

Theorem 4.2.2. The EXACT DEFENSIVE ALLIANCE problem is W[1]-hard when parameterized by the feedback vertex set number and the pathwidth of the input graph.

We introduce some variants of DEFENSIVE ALLIANCE that we require in our proofs. The problem DEFENSIVE ALLIANCE^F generalizes DEFENSIVE ALLIANCE where some vertices are forced to be outside the solution; these vertices are called “forbidden” vertices. This variant can be formalized as follows:

DEFENSIVE ALLIANCE^F
Input: An undirected graph $G = (V, E)$, an integer k and a set $V_{\square} \subseteq V(G)$.
Question: Is there a defensive alliance $S \subseteq V$ such that (i) $1 \leq |S| \leq k$, and (ii) $S \cap V_{\square} = \emptyset$?

DEFENSIVE ALLIANCE^{FN} is a further generalization that, in addition, requires some “necessary” vertices to be in S . This variant can be formalized as follows:

DEFENSIVE ALLIANCE^{FN}
Input: An undirected graph $G = (V, E)$, an integer k , a set $V_{\Delta} \subseteq V$, and a set $V_{\square} \subseteq V(G)$.
Question: Is there a defensive alliance $S \subseteq V$ such that (i) $1 \leq |S| \leq k$, (ii) $S \cap V_{\square} = \emptyset$, and (iii) $V_{\Delta} \subseteq S$?

While the DEFENSIVE ALLIANCE problem asks for defensive alliance of size at most k , we also consider the EXACT DEFENSIVE ALLIANCE problem that concerns defensive alliance of size exactly k . Analogously, we also define exact versions of the two generalizations of DEFENSIVE ALLIANCE presented above.

4.2.1 Hardness of Defensive Alliance with Forbidden and Necessary Vertices

To show $W[1]$ -hardness of $\text{DEFENSIVE ALLIANCE}^{\text{FN}}$, we consider the $\text{MULTIDIMENSIONAL SUBSET SUM}$ (MSS) problem.

MULTIDIMENSIONAL SUBSET SUM (MSS)
Input: An integer k , a set $S = \{s_1, \dots, s_n\}$ of vectors with $s_i \in \mathbb{N}^k$ for every i with $1 \leq i \leq n$ and a target vector $t \in \mathbb{N}^k$.
Parameter: k
Question: Is there a subset $S' \subseteq S$ such that $\sum_{s \in S'} s = t$?

We introduce two variants of MSS that we require in our proofs. In the $\text{MULTIDIMENSIONAL RELAXED SUBSET SUM}$ (MRSS) problem, an additional integer k' is given (which will be part of the parameter) and we ask whether there is a subset $S' \subseteq S$ with $|S'| \leq k'$ such that $\sum_{s \in S'} s \geq t$. It is known that MRSS is $W[1]$ -hard when parameterized by the combined parameter $k + k'$, even if all integers in the input are given in unary [30]. For EXACT MRSS problem, both the input as well as the parameters are the same as in the case of MRSS however one now asks whether there is a subset $S' \subseteq S$ with $|S'| = k'$ such that $\sum_{s \in S'} s \geq t$.

This variant can be formalized as

EXACT MULTIDIMENSIONAL RELAXED SUBSET SUM (EXACT MRSS)
Input: An integer k , a set $S = \{s_1, \dots, s_n\}$ of vectors with $s_i \in \mathbb{N}^k$ for every i with $1 \leq i \leq n$ and a target vector $t \in \mathbb{N}^k$.
Parameter: k, k'
Question: Is there a subset $S' \subseteq S$ with $|S'| = k'$ such that $\sum_{s \in S'} s \geq t$?

Lemma 3. EXACT MRSS is $W[1]$ -hard when parameterized by the combined parameter $k + k'$, even if all integers in the input are given in unary.

This follows from the fact that the MRSS problem is $W[1]$ -hard even if all integers in the input are given in unary. We now show that the $\text{DEFENSIVE ALLIANCE}^{\text{FN}}$ problem is

W[1]-hard parameterized by the size of a vertex deletion set into trees of height at most 4, via a reduction from MRSS.

Lemma 4. The DEFENSIVE ALLIANCE^{FN} problem is W[1]-hard when parameterized by the size of a vertex deletion set into trees of height at most 4.

Proof. To prove this we reduce from MRSS, which is known to be W[1]-hard when parameterized by the combined parameter $k + k'$ [30]. Let $I = (k, k', S, t)$ be an instance of MRSS. Let $s = (s(1), s(2), \dots, s(k)) \in S$ and let $\max(s)$ denote the value of the largest coordinate of s . We construct an instance $I' = (G, r, V_\Delta, V_\square)$ of DEFENSIVE ALLIANCE^{FN} the following way. Before we formally define our reduction, we briefly describe the intuition. We introduce three types of vertices: necessary vertices, forbidden vertices and normal vertices. We often indicate necessary vertices by means of a triangular node shape, and forbidden vertices by means of a square node shape, and normal vertices by means of a circular node shape. We want to make sure that *necessary vertices* are always inside every solution and *forbidden vertices* are always outside every solution; and a *normal vertex* could be inside or outside the solution. Note that it is easy to force a forbidden vertex outside every solution by simply increasing its degree in the graph. Specifically, if we are looking for a defensive alliance of size at most r then vertices of degree more than $2r$ are always outside every solution. The challenging part is to force inclusion of all necessary vertices in every solution. See Figure 4.1 for an illustration of the reduction. The vertex set of the constructed graph G is defined as follows:

1. We introduce a set of k new vertices $U = \{u_1, u_2, \dots, u_k\}$. For every $u_i \in U$, we create a set $V_{u_i\square}$ of $\sum_{s \in S} s(i)$ one degree forbidden vertices and a set $V_{u_i\Delta}$ of $2 \left(\sum_{s \in S} s(i) - t(i) \right)$ one degree necessary vertices.
2. For each vector $s = (s(1), s(2), \dots, s(k)) \in S$, we introduce a tree T_s into G . We introduce two vertices x_s and y_s , and introduce two sets of new vertices $A_s = \{a_1^s, \dots, a_{\max(s)}^s\}$ and $B_s = \{b_1^s, \dots, b_{\max(s)}^s\}$. Next, for every vertex $a^s \in A_s$, we add a set $V_{a^s\square}$ of $|N_U(a^s)| + 3$ many one degree forbidden vertices. Therefore, the vertex set of tree T_s is as follows:

$$V(T_s) = A_s \cup B_s \cup \bigcup_{a^s \in A_s} V_{a^s\square} \cup \{x_s, y_s\}.$$

3. Finally two vertices a, b are introduced into G .

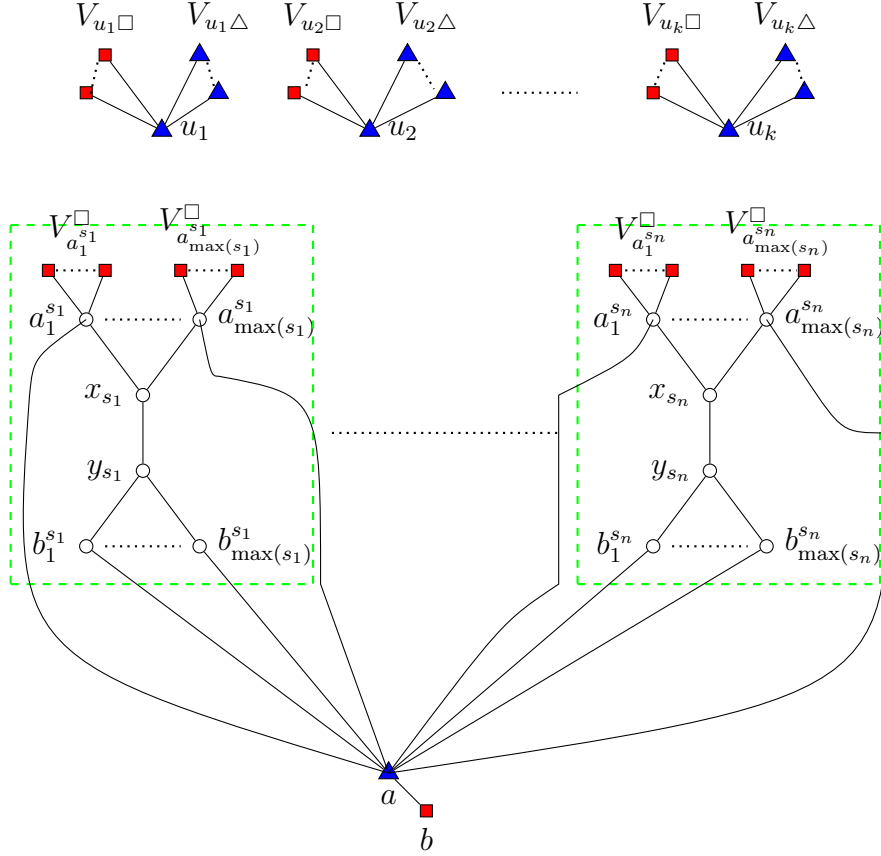


Figure 4.1: The reduction from MRSS to DEFENSIVE ALLIANCE^{FN} in Lemma 4. Note that the edges between the vertices in U and A_s are not shown. Gadgets in the green square correspond to vectors in set S .

We now create the edge set of G .

1. For every $u_i \in U$, make u_i adjacent to every vertex of $V_{u_i, \square} \cup V_{u_i, \Delta}$.
2. For every $s \in S$, we create the edge set of T_s ,

$$E(T_s) = \left\{ (x_s, \alpha), (y_s, \beta), (x_s, y_s) \mid \alpha \in A_s, \beta \in B_s \right\} \\ \cup_{a^s \in A_s} \left\{ (a^s, \alpha) \mid \alpha \in V_{a^s, \square} \right\}.$$

3. For each $i \in \{1, 2, \dots, k\}$ and for each $s \in S$, we make u_i adjacent to exactly $s(i)$ many vertices of A_s in an arbitrary manner.

4. For each $s \in S$, make a adjacent to all vertices of $A_s \cup B_s$; also make b adjacent to a .

We define the set of necessary vertices

$$V_{\Delta} = \{a\} \cup U \cup \bigcup_{i=1}^k V_{u_i \Delta},$$

the set of forbidden vertices

$$V_{\square} = \{b\} \cup \bigcup_{s \in S} \bigcup_{a^s \in A_s} V_{a^s}^{\square},$$

and set $r = \sum_{i=1}^k 2 \left(\sum_{s \in S} s(i) - t(i) \right) + \sum_{i=1}^n \max(s_i) + k + k' + 1$. Observe that if we remove the set $U \cup \{a\}$ of $k + 1$ vertices from G , each connected component of the resulting graph is a tree with height at most 4.

Formally, we claim (k, k', S, t) is a positive instance of MRSS if and only if G has a defensive alliance S of size at most r such that $V_{\Delta} \subseteq R$, and $V_{\square} \cap R = \emptyset$. Let $S' \subseteq S$ be such that $|S'| \leq k'$ and $\sum_{s \in S'} s \geq t$. We claim that the set

$$\begin{aligned} R &= V_{\Delta} \cup \bigcup_{s \in S'} (A_s \cup \{x_s\}) \cup \bigcup_{s \in S \setminus S'} B_s \\ &= \{a\} \cup U \cup \bigcup_{i=1}^k V_{u_i \Delta} \cup \bigcup_{s \in S'} (A_s \cup \{x_s\}) \cup \bigcup_{s \in S \setminus S'} B_s \end{aligned}$$

is a defensive alliance in G such that $|R| \leq r$, $V_{\Delta} \subseteq R$, and $V_{\square} \cap R = \emptyset$. Let x be an arbitrary element of R .

Case 1: If $x = u_i \in U$, then the neighbours of u_i in R are the elements of $V_{u_i \Delta}$ and $s(i)$ elements of A_s if $s \in S'$. Therefore we have

$$d_R(u_i) = \sum_{s \in S'} s(i) + |V_{u_i \Delta}|.$$

The neighbours of u_i in R^c are the elements of $V_{u_i}^{\square}$ and $s(i)$ elements of A_s if $s \in S \setminus S'$. Thus

$$d_{R^c}(u_i) = \sum_{s \in S \setminus S'} s(i) + |V_{u_i}^{\square}| = \sum_{s \in S \setminus S'} s(i) + \sum_{s \in S} s(i).$$

Note that

$$\begin{aligned}
d_R(u_i) &= \sum_{s \in S'} s(i) + |V_{u_i \Delta}| \\
&= \sum_{s \in S'} s(i) + 2 \sum_{s \in S} s(i) - 2t(i) \\
&= \underbrace{\left(\sum_{s \in S'} s(i) - t(i) \right)}_{\geq 0 \text{ by assumption}} + \left(\sum_{s \in S} s(i) - t(i) \right) + \sum_{s \in S} s(i) \\
&\geq \left(\sum_{s \in S} s(i) - t(i) \right) + \sum_{s \in S} s(i) \\
&= \sum_{s \in S \setminus S'} s(i) + \underbrace{\left(\sum_{s \in S'} s(i) - t(i) \right)}_{\geq 0 \text{ by assumption}} + \sum_{s \in S} s(i) \\
&\geq \sum_{s \in S \setminus S'} s(i) + \sum_{s \in S} s(i) = \sum_{s \in S \setminus S'} s(i) + |V_{u_i \square}| \\
&= d_{R^c}(u_i)
\end{aligned}$$

Therefore, we have $d_R(u_i) + 1 \geq d_{R^c}(u_i)$, and hence u_i is protected.

Case 2: If $x = a^s \in A_s$, then $d_R(a^s) = |N_U(a^s)| + |\{a, x_s\}| = |N_U(a^s)| + 2$ and $d_{R^c}(a^s) = |V_{a^s \square}| = |N_U(a^s)| + 3$. Therefore, we have $d_R(a^s) + 1 \geq d_{R^c}(a^s)$.

Case 3: If $x = b^s \in B_s$, then $N_R(b^s) = \{a\}$ and $N_{R^c}(b^s) = \{y_s\}$. Therefore, we have $d_R(b^s) + 1 \geq d_{R^c}(b^s)$.

Case 4: If $x = a$, then $N_R(a) = \bigcup_{s \in S'} A_s \cup \bigcup_{s \in S \setminus S'} B_s$ and $N_{R^c}(a) = \bigcup_{s \in S'} B_s \cup \bigcup_{s \in S \setminus S'} A_s \cup \{b\}$. As $|A_s| = |B_s|$, we have $d_R(a) + 1 \geq d_{R^c}(a)$.

For the vertices in $V_{u\Delta}$, it is easy to see that they have one neighbour in R and no neighbours in R^c . Therefore, R is a defensive alliance in G such that $|R| \leq r$, $V_{\Delta} \subseteq R$, and $V_{\square} \cap R = \emptyset$.

For the reverse direction, suppose that G has a defensive alliance R of size at most r such

that $V_\Delta \subseteq R$ and $V_\square \cap R = \emptyset$. From the definition of V_Δ , we have $U \subseteq R$. We know V_Δ contains $1 + k + \sum_{i=1}^k 2\left(\sum_{s \in S} s(i) - t(i)\right)$ vertices; thus besides the vertices of V_Δ , there are at most $\sum_{i=1}^n \max(s_i) + k'$ vertices in R . The neighbours of a in G are the elements of $\bigcup_{s \in S} A_s \cup B_s$; therefore the degree of a in G is $\sum_{i=1}^n 2 \times \max(s_i)$. Since $a \in R$, it is protected and it must have at least $\sum_{i=1}^n \max(s_i)$ many neighbours in R from the set $\bigcup_{s \in S} A_s \cup B_s$. For each $a^s \in A_s$, we have $N_G(a^s) = V_{a^s}^\square \cup N_U(a^s) \cup \{a, x_s\}$. As the elements of $V_{a^s}^\square$ are forbidden vertices and $|V_{a^s}^\square| = |N_U(a^s)| + 3$, a^s has at least $|N_U(a^s)| + 3$ neighbours outside R . Therefore if $a^s \in A_s$ is in the solution then x_s also lie in the solution for the protection of a^s . This shows that at most k' many sets of the form A_s contribute to the solution as otherwise the size of solution exceeds r . Therefore, any arbitrary defensive alliance R of size at most r can be transformed to another defensive alliance R' of size at most r as follows:

$$R' = V_\Delta \bigcup_{x_s \in R} A_s \cup \{x_s\} \bigcup_{x_s \in V(G) \setminus R} B_s.$$

We define a subset $S' = \left\{s \in S \mid x_s \in R'\right\}$. Clearly, $|S'| \leq k'$. We claim that $\sum_{s \in S'} s(i) \geq t(i)$ for all $1 \leq i \leq k$. Assume for the sake of contradiction that $\sum_{s \in S'} s(i) < t(i)$ for some

$i \in \{1, 2, \dots, k\}$. Then, we have

$$\begin{aligned}
d_{R'}(u_i) &= \sum_{s \in S'} s(i) + |V_{u_i \Delta}| \\
&= \sum_{s \in S'} s(i) + \underbrace{2 \sum_{s \in S} s(i) - 2t(i)}_{\text{size of } V_{u_i \Delta}} \\
&= \underbrace{\sum_{s \in S'} s(i) - t(i)}_{<0 \text{ by assumption}} + \sum_{s \in S} s(i) - t(i) + \sum_{s \in S} s(i) \\
&< \sum_{s \in S} s(i) - t(i) + \sum_{s \in S} s(i) \\
&= \sum_{s \in S \setminus S'} s(i) + \underbrace{\left(\sum_{s \in S'} s(i) - t(i) \right)}_{<0 \text{ by assumption}} + \sum_{s \in S} s(i) \\
&< \sum_{s \in S \setminus S'} s(i) + \sum_{s \in S} s(i) = \sum_{s \in S \setminus S'} s(i) + |V_{u_i \square}| \\
&= d_{R'^c}(u_i)
\end{aligned}$$

and we also know that $u_i \in R'$, which is a contradiction to the fact that R' is a defensive alliance. Therefore we get that $\sum_{s \in S'} s(i) \geq t(i)$ for all $1 \leq i \leq k$. This shows that (k, k', S, t) is a positive instance of MRSS. \square

The following corollary is a consequence of Lemma 4.

Corollary 1. The DEFENSIVE ALLIANCE^{FN} problem is W[1]-hard when parameterized by the size of a vertex deletion set into trees of height at most 5, even when $|V_\Delta| = 1$.

Proof. Let $I = (G, r, V_\Delta, V_\square)$ be an instance of DEFENSIVE ALLIANCE^{FN} where $V_\Delta = \{v_1, v_2, \dots, v_\ell\}$, $\ell > 1$. We construct an equivalent instance $I' = (G', r', V'_\Delta, V'_\square)$ of DEFENSIVE ALLIANCE^{FN} where $|V'_\Delta| = 1$. The construction of G' starts with $G' := G$ and then add the following new vertices. We introduce a necessary vertex x and make x adjacent to all the vertices in V_Δ . We introduce a set $V_{x\square}$ of $\ell + 1$ degree one forbidden vertices and make x adjacent to every vertex of $V_{x\square}$. For every $v_i \in V_\Delta$, add a degree one forbidden vertex v_i^\square and make it adjacent to v_i . We define $V'_\Delta = \{x\}$ and $V'_\square = V_{x\square} \cup V_\square \cup \bigcup_{v_i \in V_\Delta} \{v_i^\square\}$. We define

G' as follows

$$V(G') = V(G) \cup \{x\} \cup V_{x\Box} \cup \bigcup_{v_i \in V_\Delta} v_i^\Box$$

and

$$E(G') = E(G) \cup \{(x, \alpha), (x, v) \mid \alpha \in V_{x\Box}, v \in V_\Delta\} \cup \bigcup_{i=1}^{\ell} \{(v_i, v_i^\Box)\}.$$

We set $r' = r + 1$.

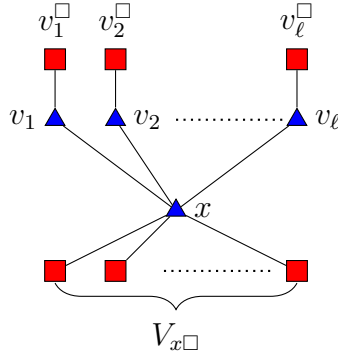


Figure 4.2: An illustration of the reduction in Corollary 1.

Let H be a set with at most k vertices in G such that $G - H$ is a forest with trees of height at most 4. Clearly, the set $H \cup \{x\}$ is of size at most $k + 1$ and $G' - (H \cup \{x\})$ is a forest with trees of height at most 5. We claim that G has a defensive alliance S of size at most r such that $V_\Delta \subseteq S$ and $V_\Box \cap S = \emptyset$ if and only if G' has a defensive alliance S' of size at most r' such that $V'_\Delta \subseteq S'$ and $V'_\Box \cap S' = \emptyset$. Suppose that G has a defensive alliance S of size at most r such that $V_\Delta \subseteq S$ and $V_\Box \cap S = \emptyset$. Define $S' = S \cup \{x\}$. Note that $N_{S'}(x) = \{v_1, v_2, \dots, v_\ell\}$ and the neighbour of x outside S' are the elements of $V_{x\Box}$. That is, x has $\ell + 1$ neighbours (including itself) in S' and $\ell + 1$ neighbours outside S' . So x is protected in S' . Clearly, every vertex of S remains protected. Therefore S' is a defensive alliance of size at most r such that $V'_\Delta \subseteq S'$ and $V'_\Box \cap S' = \emptyset$.

To prove the reverse direction of the equivalence, suppose now that S' is a defensive alliance of size at most r' in G' such that $V'_\Delta \subseteq S'$ and $V'_\Box \cap S' = \emptyset$. The protection of x in S' ensures that $\{v_1, v_2, \dots, v_\ell\} \subseteq S'$. Define $S = S' \setminus \{x\}$. Clearly, S is of size at most r . As $\{v_1, v_2, \dots, v_\ell\} \subseteq S'$, we have $V_\Delta = \{v_1, v_2, \dots, v_\ell\} \subseteq S$. Observe that $N_S(v_i) = N_{S'}(v_i) \setminus \{x\}$

and $N_{S^c}(v_i) = N_{S^c}(v_i) \setminus \{v_i^\square\}$. As v_i was protected in S' of G' , it remains protected in S of G . Thus S is a defensive alliance in G of size at most r such that $V_\Delta \subseteq S$ and $V_\square \cap S = \emptyset$. \square

We can get an analogous result for the exact variant.

Corollary 2. The EXACT DEFENSIVE ALLIANCE^{FN} problem is W[1]-hard when parameterized by the size of a vertex deletion set into trees of height at most 5, even when $|V_\Delta| = 1$.

The following corollary is a consequence of Lemma 4.

Corollary 3. The DEFENSIVE ALLIANCE^{FN} problem is W[1]-hard when parameterized by pathwidth and treedepth of the input graph.

Proof. To prove this we reduce from MRSS, which is known to be W[1]-hard when parameterized by the combined parameter $k + k'$ [30]. Let $I = (k, k', S, t)$ be an instance of MRSS. We construct an instance $I' = (G, r, V_\Delta, V_\square)$ of DEFENSIVE ALLIANCE^{FN} as in Lemma 4. Note that the pathwidth and treedepth of a tree are at most its height. We claim that as G has a $k + 1$ size vertex deletion set $D = U \cup \{a\}$ into trees of height at most 4, then G has a path decomposition of width at most $k + 4$. First, we get a path decomposition \mathcal{P} of trees of height at most 4, it has width at most 3; then add $k + 1$ vertices of D to all the bags of \mathcal{P} to get a path decomposition of G . This implies that the pathwidth of G is bounded by $k + 4$. To compute treedepth of G , note that $G \setminus D$ is a rooted forest where the trees are of height at most 4. We add paths of length at most $k + 1$ at the roots, covering the vertices of D , such that the resulting forest is a transitive closure of G . The height of the resulting forest is at most $k + 5$. This implies that the treedepth of G is bounded by $k + 5$. \square

4.2.2 Hardness of Defensive Alliance with Forbidden Vertices

Now we give an FPT reduction to get rid of necessary vertices for the exact version of the problem.

Lemma 5. The EXACT DEFENSIVE ALLIANCE^F problem is W[1]-hard parameterized by the size of a vertex deletion set into trees of height at most 5.

Proof. To prove this we reduce from EXACT DEFENSIVE ALLIANCE^{FN} when $|V_\Delta| = 1$, which is W[1]-hard when parameterized by the size of a vertex deletion set into trees of height at most 5. See Corollary 2. Let $I = (G, r, V_\Delta, V_\square)$ with $|V_\Delta| = 1$ be an instance of EXACT DEFENSIVE ALLIANCE^{FN}. Let $n = |V(G)|$, $r \leq n$ and $V_\Delta = \{x\}$. We construct an instance $I' = (G', r', V'_\square)$ of EXACT DEFENSIVE ALLIANCE^F problem the following way. See Figure 4.3 for an illustration. The construction of G' starts with $G' := G$ and then add

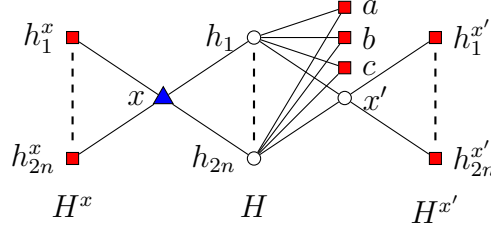


Figure 4.3: An illustration of the reduction from EXACT DEFENSIVE ALLIANCE^{FN} with $|V_\Delta| = 1$ to EXACT DEFENSIVE ALLIANCE^F. The vertex x may have additional neighbours in G .

the following new vertices and edges. We introduce three forbidden vertices a, b, c and one normal vertex x' . We also introduce a set of $2n$ normal vertices $H = \{h_1, h_2, \dots, h_{2n}\}$, two sets of $2n$ forbidden vertices each: $H^x = \{h_1^x, h_2^x, \dots, h_{2n}^x\}$ and $H^{x'} = \{h_1^{x'}, h_2^{x'}, \dots, h_{2n}^{x'}\}$. We make x adjacent to every vertex of $H \cup H^x$ and make x' adjacent to every vertex of $H \cup H^{x'}$. Finally make a, b, c adjacent to every vertex of H . We define G' as follows:

$$V(G') = V(G) \cup H \cup H^x \cup H^{x'} \cup \{a, b, c, x'\}$$

and

$$E(G') = E(G) \cup \bigcup_{h \in H} \{(x, h), (x', h), (a, h), (b, h), (c, h)\} \cup \bigcup_{h^x \in H^x} (x, h^x) \cup \bigcup_{h^{x'} \in H^{x'}} (x', h^{x'}).$$

We define $V'_\square = V_\square \cup H^x \cup H^{x'} \cup \{a, b, c\}$ and set $r' = r + 2n + 1$. Suppose G has a size k vertex deletion set D to trees of height at most 4. Then $D \cup \{x, x', a, b, c\}$ is a vertex deletion set to trees of height at most 4 in G' . Thus the size of a vertex deletion set to trees of height at most 4 is clearly bounded by $k + 5$ in G' .

We claim that I is a yes-instance if and only if I' is a yes-instance. Suppose there is a

defensive alliance R of size exactly r in G such that $x \in R$ and $V_{\square} \cap R = \emptyset$. It is easy to check that $R' = R \cup H \cup \{x'\}$ is a defensive alliance of size $r + 2n + 1$ such that $V'_{\square} \cap R' = \emptyset$. This implies that I' is a yes-instance.

To prove the reverse direction of the equivalence, suppose there is a defensive alliance R' of size r' such that $V'_{\square} \cap R' = \emptyset$. We claim that $x \in R'$. For the sake of contradiction, suppose $x \notin R'$. Then no vertex from the set H is part of R' as $d_{R'}(h) \leq 2$ and $d_{R^c}(h) \geq 4$ for each $h \in H$. This implies that $|R'| \leq n < r'$, a contradiction. Therefore $x \in R'$. Observe that at least one vertex from H must be part of R' for protection of x in R' . Without loss of generality assume that $h_1 \in R'$. We see that the protection of h_1 requires x' to be inside the solution. Therefore, x' is in R' . Now, the protection of x' requires $2n$ many vertices which can only be contributed by H . It implies that $H \subseteq R'$. We claim that $R = R' \setminus (H \cup \{x\})$ forms a defensive alliance of size exactly r in G . Since $R' \cap V(G) = \{x\}$, we only need to show that x is protected R . This is true since x loses $2n$ neighbours from inside and outside the solution in G' . This shows that I is a yes-instance. \square

In [6], Bliem and Woltran proved that $\text{DEFENSIVE ALLIANCE}^{\text{F}}$ problem is $\text{W}[1]$ -hard when parameterized by the treewidth of the input graph by giving a reduction from $\text{DEFENSIVE ALLIANCE}^{\text{FN}}$ problem which again they showed to be $\text{W}[1]$ -hard when parameterized by the treewidth of the graph. Since we have a stronger result that the $\text{DEFENSIVE ALLIANCE}^{\text{FN}}$ is $\text{W}[1]$ -hard when parameterized by the pathwidth of the input graph, we now prove that the $\text{DEFENSIVE ALLIANCE}^{\text{F}}$ problem is $\text{W}[1]$ -hard when parameterized by the pathwidth of the input graph. We obtain the following hardness results using the reduction of Lemma 5 given in [6] and see that the resulting graph has bounded pathwidth.

Lemma 6. The $\text{DEFENSIVE ALLIANCE}^{\text{F}}$ problem is $\text{W}[1]$ -hard when parameterized by the pathwidth of the input graph.

Proof. To prove this we reduce from $\text{DEFENSIVE ALLIANCE}^{\text{FN}}$, which is $\text{W}[1]$ -hard when parameterized by the pathwidth of the input graph. See Corollary 3. The reduction given here is the same as that of Lemma 5 in [6]. Let $I = (G, r, V_{\Delta}, V_{\square})$ be an instance of $\text{DEFENSIVE ALLIANCE}^{\text{FN}}$. We construct an instance $I' = (G', r', V'_{\square})$ of $\text{DEFENSIVE ALLIANCE}^{\text{F}}$ problem in the following way. First, we find an optimal path decomposition \mathcal{P} of G which can be done in FPT time. Next, we record the sequence \preceq of non-forbidden vertices that occur for the last time in \mathcal{P} . Let V_0 denote $V(G) \setminus (V_{\Delta} \cup V_{\square})$. For each $v \in V_0$, we introduce a set

of new vertices $H = \{v', g_v, h_v, g_v^\square, h_v^\square\}$. We use V^+ to denote the set $V_\Delta \cup V_0 \cup \{v' \mid v \in V_0\}$. For each $v \in V^+$, we introduce the set of vertices $A_v = \{v_1, \dots, v_{n+1}, v_1^\square, \dots, v_{n+1}^\square\}$; we use the notation $A_v^\square = \{v_1^\square, \dots, v_{n+1}^\square\}$. We use the notation $u \oplus v$ to denote the set of edges $\{(u, v), (u, u^\square), (v, v^\square), (u, v^\square), (v, u^\square)\}$. For any vertex $v \in V_0 \cup V_\Delta$, we define $p(v) = v$ if $v \in V_\Delta$ and $p(v) = v'$ if $v \in V_0$. Let P be the set consisting of all pairs $(p(u), p(v))$ such that v is the direct successor of u in the sequence \preceq . Next, we define $V'_\square = V_\square \cup \{g_v^\square, h_v^\square \mid v \in V_0\} \cup \bigcup_{v \in V^+} A_v^\square$. The graph G' is defined by

$$V(G') = V(G) \cup H \cup \bigcup_{v \in V^+} A_v$$

and

$$\begin{aligned} E(G') &= E(G) \cup \{(v, v_i), (v, v_i^\square) \mid v \in V^+, 1 \leq i \leq n+1\} \\ &\cup \bigcup_{v \in V^+, 1 \leq i \leq n} v_i \oplus v_{i+1} \cup \bigcup_{(u,v) \in P} u_{n+1} \oplus v_1 \\ &\cup \bigcup_{v \in V_0} v_{n+1} \oplus g_v \cup \{(v', g_v), (v', h_v), (g_v, h_v), (g_v, h_v^\square) \mid v \in V_0\}. \end{aligned}$$

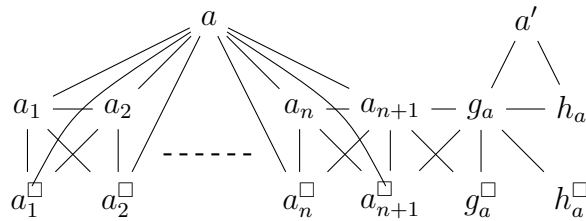


Figure 4.4: An illustration of the gadget corresponding to a non-necessary and non-forbidden vertex.

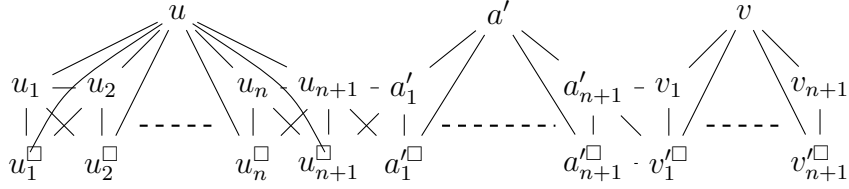


Figure 4.5: Illustration of the gadget that makes sure that every solution contains all necessary vertices if it contains some necessary vertex or if it contains a' for some non-necessary vertex a . We use the ordering $u \preceq a \preceq v$ where u, v are necessary vertices and a is non-necessary and non-forbidden vertex.

See Figure 4.4 and 4.5 for an illustration. Next, we show that the pathwidth of G' is bounded by a function of the pathwidth of G . To get a path decomposition \mathcal{P}' of G' , we use an optimal path decomposition \mathcal{P} of G . In the path decomposition \mathcal{P} , we denote the right most bag containing vertex u by $t_u^{\mathcal{P}}$. Initially, we set $\mathcal{P}' := \mathcal{P}$ and make the following modifications:

- For each $v \in V_0$, we add the vertices $v', g_v, h_v, g_v^\square, h_v^\square$ to the bag of $t_v^{\mathcal{P}'}$. After this step the pathwidth of \mathcal{P}' is increased by at most 5.
- For each $v \in V^+$, let B_v denote the bag of $t_v^{\mathcal{P}'}$ and replace $t_v^{\mathcal{P}'}$ by a sequence of nodes N_1, \dots, N_n where N_n is the rightmost node and the bag of N_i is $B_v \cup \{v_i, v_{i+1}, v_i^\square, v_{i+1}^\square\}$. Note that we have covered all the edges except the ones connecting the nodes of two distinct sets A_u and A_v where $(u, v) \in P$. After this step, the pathwidth of \mathcal{P}' is increased by at most nine.
- For every $(u, v) \in P$, we add v_1 and v_1^\square into the bag of every node between (and including) $t_u^{\mathcal{P}'}$ and $t_v^{\mathcal{P}'}$. In this way, we can cover all the remaining edges. This implies that the pathwidth \mathcal{P}' of G' is at most pathwidth \mathcal{P} of G plus 11.

The proof of equivalence of instances I and I' is given in [6].

4.2.3 Hardness of Defensive Alliance

Now we give an FPT reduction to get rid of forbidden vertices. The same reduction holds for exact version of the problem as well.

Proof of Theorem 4.2.1

Proof. To prove Theorem 4.2.1 we reduce from the DEFENSIVE ALLIANCE^F problem, which is W[1]-hard when parameterized by the pathwidth of the input graph. See Lemma 6. Let $I = (G, r, V_\square)$ be an instance of the DEFENSIVE ALLIANCE^F problem. We construct an instance $I' = (G', r')$ of DEFENSIVE ALLIANCE problem the following way. We set $r' = r$.

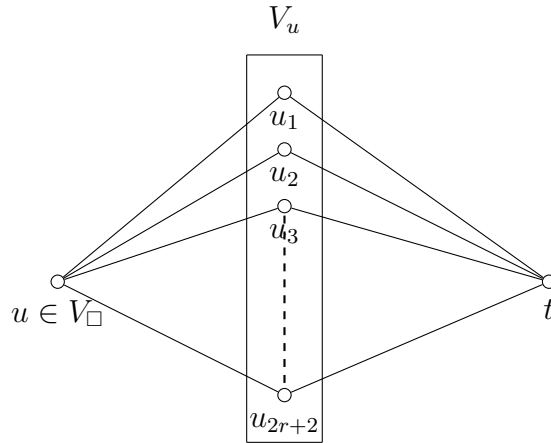


Figure 4.6: An illustration of the reduction from DEFENSIVE ALLIANCE^F to DEFENSIVE ALLIANCE.

For every vertex $u \in V_\square$, we introduce a set $V_u = \{u_1, u_2, \dots, u_{2r+2}\}$ of $2r + 2$ many vertices and make them adjacent to u . We also add a new vertex t which is adjacent to all the vertices in $\bigcup_{u \in V_\square} V_u$. Clearly, we can see that the pathwidth of G' is at most the pathwidth of G plus two. We claim that I is a yes-instance if and only if I' is a yes-instance. Suppose there is a defensive alliance R of size at most r in G such that $V_\square \cap R = \emptyset$. Clearly $R' = R$ is also a defensive alliance of size at most $r' = r$ in G' . This implies that I' is a yes-instance.

To prove the reverse direction of the equivalence, suppose there is a defensive alliance R' in G' of size at most $r' = r$. We claim that any defensive alliance in G' containing a vertex

from the set $V_{\square} \cup \bigcup_{u \in V_{\square}} V_u \cup \{t\}$ is of size at least $r + 1$. The reason is this. Let x be an arbitrary element of $V_{\square} \cup \bigcup_{u \in V_{\square}} V_u \cup \{t\}$. If x is an element of V_{\square} then $d_{G'}(x) \geq 2r + 2$. Therefore its protection requires at least $r + 1$ vertices in R' . If $x = t$, then $d_{G'}(t) = |V_{\square}|(2r + 2) \geq (2r + 2)$. Therefore its protection requires at least $r + 1$ vertices in R' . Since $|R'| \leq r' = r$, this implies that $R' \cap (V_{\square} \cup \{t\}) = \emptyset$. If $x \in V_u$ for some $u \in V_{\square}$, then x has two neighbours u and t , and we proved that both u and t are not in R' . Therefore x also cannot be in R' . This implies that $R' \cap (V_{\square} \cup \bigcup_{u \in V_{\square}} V_u \cup \{t\}) = \emptyset$. We see that $R = R'$ is a defensive alliance of size at most r such that $R \cap V_{\square} = \emptyset$. This shows that I is a yes-instance. \square

The following corollary is a consequence of Theorem 4.2.1.

Corollary 4. The DEFENSIVE ALLIANCE^N problem with exactly one necessary vertex is W[1]-hard when parameterized by the size of a vertex deletion set into trees of height at most 6.

Proof. To prove this we reduce from DEFENSIVE ALLIANCE^{FN} problem with $|V_{\Delta}| = 1$, which is W[1]-hard when parameterized by the size of a vertex deletion set into trees of height at most 5. See Corollary 1. The reduction here is the same as in the proof of Theorem 4.2.1. \square

Proof of Theorem 4.2.2

Proof. To prove Theorem 4.2.2 we reduce from EXACT DEFENSIVE ALLIANCE^F problem, which is W[1]-hard when parameterized by the vertex deletion set into trees of height at most 4 of the input graph. See Lemma 5. The reduction here is the same as in the proof of Theorem 4.2.1. Therefore, the EXACT DEFENSIVE ALLIANCE problem is W[1]-hard when parameterized by the vertex deletion set into trees of height at most 6. Clearly trees of height at most 6 are trivially acyclic. Moreover, it is easy to verify that such trees have pathwidth [44] treedepth [49] at most 6, which implies the EXACT DEFENSIVE ALLIANCE problem is W[1]-hard when parameterized by any of the following parameters: the feedback vertex set, pathwidth, treewidth and treedepth of the input graph. \square

Chapter 5

Locally Minimal Defensive Alliance

5.1 Introduction

An alliance S is called a *locally minimal alliance* if for any $v \in S$, $S \setminus \{v\}$ is not an alliance. It is important to note that if S is a locally minimal defensive alliance, then for every vertex $v \in S$, at least one of its neighbours in S is marginally protected. In this chapter, we consider the problem LOCALLY MINIMAL DEFENSIVE ALLIANCE, which takes as input an undirected graph G and a positive integer k , and the objective is to decide if G has a locally minimal defensive alliance of size at least k . This chapter is based on the paper [26] and the manuscript [27]. We design a polynomial-time algorithm for the CONNECTED LOCALLY MINIMAL STRONG DEFENSIVE ALLIANCE on trees. We prove that LOCALLY MINIMAL DEFENSIVE ALLIANCE problem is NP-complete, even when restricted to planar graphs. We give a randomized FPT algorithm for the EXACT CONNECTED LOCALLY MINIMAL DEFENSIVE ALLIANCE problem using color coding technique. We give an FPT algorithm for LOCALLY MINIMAL DEFENSIVE ALLIANCE when parameterized by neighbourhood diversity of the input graph. We prove that EXACT CONNECTED LOCALLY MINIMAL DEFENSIVE ALLIANCE parameterized by treewidth is W[1]-hard and thus not FPT (unless FPT=W[1]). Finally we show that the LOCALLY MINIMAL DEFENSIVE ALLIANCE problem is polynomial time solvable for graphs with bounded treewidth.

5.2 Polynomial-Time Algorithm for Connected Locally Minimal Strong Defensive Alliance on Trees

Finding the largest connected locally minimal (strong) defensive alliance is believed to be intractable [1]. However, when the graph happens to be a tree, we solve the problem in polynomial time, using dynamic programming. Here we consider strong defensive alliances where the vertex itself is not counted while computing the number of neighbours in the defensive alliance. A defensive alliance is *strong* if each vertex in the alliance has at least as many neighbours in the alliance (not counting itself) as outside the alliance. A vertex v in the strong defensive alliance S is said to be *marginally protected* if it becomes unprotected when any of its neighbour in S is moved from S to $V \setminus S$. A vertex $v \in S$ is said to be *overprotected* if it remains protected even when any of its neighbours is moved from S to $V \setminus S$. Also recall that if S is a locally minimal (strong) defensive alliance, then for every vertex $v \in S$, at least one of its neighbours in S is marginally protected. A vertex $v \in S$ is said to be *good* if it has at least one marginally protected children in S , otherwise it is called a *bad* vertex. Let v be a non-root node with children v_1, v_2, \dots, v_d , that is, $d(v) = d + 1$. We define different possible states of a vertex v based on four factors:

- whether the vertex is part of the solution or not

If the vertex is in the solution, then we consider the following factors:

- whether the vertex is marginally protected or overprotected
- whether the vertex is good or bad
- whether the parent of v is in the solution or not

The states are as follows:

- 0: vertex v is not in the solution.
- m_g^p : vertex v is in the solution, it is marginally protected by its children and parent, parent of v is in the solution and it has at least one marginally protected child.

- m_b^p : vertex v is in the solution, it is marginally protected by its children and parent, parent of v is in the solution and it has no marginally protected children.
- $m_g^{\bar{p}}$: vertex v is in the solution, it is marginally protected by its children only, parent of v is not in the solution and it has at least one marginally protected child.
- o_g^p : vertex v is in the solution, it is overprotected by its children and parent, parent of v is in the solution and it has at least one marginally protected child.
- o_b^p : vertex v is in the solution, it is overprotected by its children and parent, parent of v is in the solution and it has no marginally protected children.
- $o_g^{\bar{p}}$: vertex v is in the solution, it is overprotected by its children only, parent of v is not in the solution and it has at least one marginally protected child.

Note that $m_b^{\bar{p}}$ or $o_b^{\bar{p}}$ are not valid state of a vertex v as the vertex will not have any marginally protected neighbours. Here is the algorithm: Start by rooting the tree at any node v . Each node defines a subtree, the one hanging from it. This immediately suggests subproblems: $A_v(s)$ = the size of the largest connected locally minimal strong defensive alliance of the subtree rooted at v and the state of v is s ; if no connected locally minimal strong defensive alliance exists, we put $A_v(s) = -\infty$. Our final goal is to compute $\max \left\{ A_r(0), A_r(m_g^{\bar{p}}), A_r(o_g^{\bar{p}}) \right\}$ where r is the root of T .

Leaf Node: For a leaf node v , we have $A_v(0) = 0$. Note that as v does not have any children, we get $A_v(m_g^p) = -\infty, A_v(o_g^p) = -\infty, A_v(o_b^{\bar{p}}) = -\infty, A_v(m_g^{\bar{p}}) = -\infty$. As v is a one degree vertex, it cannot be overprotected in the solution, therefore, we get $A_v(o_b^p) = -\infty$. We have $A_v(m_b^p) = 1$ as this is a valid state.

Non-leaf Node: Let v be a non-leaf node with the set $\mathcal{C} = \{v_1, v_2, \dots, v_d\}$ of children. Suppose we know $A_{v_i}(s)$ for all children v_i of v . How can we compute $A_v(s)$? We now consider the following cases:

Case 1: Let the state of v be 0, that is, v is not included in the solution. Therefore, we move

on to v 's children and consider the best of their solutions. Then

$$A_v(0) = \max_{x \in \mathcal{C}} \left\{ A_x(0), A_x(m_g^{\bar{p}}), A_x(o_g^{\bar{p}}) \right\}.$$

Case 2: Let the state of v be m_g^p or m_b^p . Here v is a good vertex and therefore, must have a marginally protected child in S . Therefore, at least one child v_i must have label m_g^p or m_b^p . Next, let (v_1, v_2, \dots, v_d) be a descending ordering of \mathcal{C} according to values $\max\{A_{v_i}(m_g^p), A_{v_i}(m_b^p), A_{v_i}(o_g^p), A_{v_i}(o_b^p)\}$, that is,

$$\max\{A_{v_1}(m_g^p), A_{v_1}(m_b^p), A_{v_1}(o_g^p), A_{v_1}(o_b^p)\} \geq \dots \geq \max\{A_{v_d}(m_g^p), A_{v_d}(m_b^p), A_{v_d}(o_g^p), A_{v_d}(o_b^p)\}.$$

Let $\mathcal{C}_{k,i}$ be the set of first k children from the ordering (v_1, v_2, \dots, v_d) except vertex v_i . Thus, we have

$$A_v(m_g^p) = 1 + \max_{v_i \in \mathcal{C}} \left\{ \max\{A_{v_i}(m_g^p), A_{v_i}(m_b^p)\} + \sum_{x \in \mathcal{C}_{\lceil \frac{d+1}{2} \rceil - 2, i}} \max\{A_x(m_g^p), A_x(m_b^p), A_x(o_g^p), A_x(o_b^p)\} \right\},$$

In the case $d = 1$, we take $A_v(m_g^p) = -\infty$ as the state is invalid.

Suppose the state of v is $m_g^{\bar{p}}$, that is, v is marginally protected by its children only and it has a marginally protected children. Since the parent of v is not there in the solution, we have

$$A_v(m_g^{\bar{p}}) = 1 + \max_{v_i \in \mathcal{C}} \left\{ \max\{A_{v_i}(m_g^p), A_{v_i}(m_b^p)\} + \sum_{x \in \mathcal{C}_{\lceil \frac{d+1}{2} \rceil - 1, i}} \max\{A_x(m_g^p), A_x(m_b^p), A_x(o_g^p), A_x(o_b^p)\} \right\}$$

Case 3: Let the state of v be o_g^p . When a vertex is overprotected, all its children in the solution must be good. If any of them is bad, it remains bad, as its parent is an overprotected node.

Here v is a good vertex and therefore, must have a marginally protected child in S . As $v \in S$ is overprotected in the presence of its parent, v must have at least $\lceil \frac{d+1}{2} \rceil$ children in S , all these children need to be good. Let (v_1, v_2, \dots, v_d) be a descending ordering of \mathcal{C} according to values $\max\{A_{v_i}(m_g^p), A_{v_i}(o_g^p)\}$. Let $\mathcal{C}_{k,i}$ be the set of first k children from the ordering (v_1, v_2, \dots, v_d) except vertex v_i . We have the following recurrence relation:

$$A_v(o_g^p) = 1 + \max_{v_i \in \mathcal{C}} \left\{ A_{v_i}(m_g^p) + \sum_{x \in \mathcal{C}_{\lceil \frac{d+1}{2} \rceil - 1, i}} \max\{A_x(m_g^p), A_x(o_g^p)\} \right. \\ \left. + \sum_{x \in \mathcal{C} \setminus (\mathcal{C}_{\lceil \frac{d+1}{2} \rceil - 1, i} \cup \{v_i\})} \max(0, A_x(m_g^p), A_x(o_g^p)) \right\}.$$

Let the state of v be \bar{o}_g^p . Then we have the following recurrence relation:

$$A_v(\bar{o}_g^p) = 1 + \max_{v_i \in \mathcal{C}} \left\{ A_{v_i}(m_g^p) + \sum_{x \in \mathcal{C}_{\lceil \frac{d+1}{2} \rceil, i}} \max\{A_x(m_g^p), A_x(o_g^p)\} \right. \\ \left. + \sum_{x \in \mathcal{C} \setminus (\mathcal{C}_{\lceil \frac{d+1}{2} \rceil, i} \cup \{v_i\})} \max(0, A_x(m_g^p), A_x(o_g^p)) \right\}.$$

Case 4: Let the state of v be o_b^p . Here v is a bad vertex, that is, v has no marginally protected children in S . In other words, v 's children in S must be overprotected. As $v \in S$ is overprotected, all its children in S must be good. Let (v_1, v_2, \dots, v_d) be a descending ordering of \mathcal{C} according to values $A_{v_i}(o_g^p)$. Let $\mathcal{C}_{\lceil \frac{d+1}{2} \rceil} = \{v_1, \dots, v_{\lceil \frac{d+1}{2} \rceil}\}$. We have the following recurrence relation:

$$A_v(o_b^p) = 1 + \sum_{x \in \mathcal{C}_{\lceil \frac{d+1}{2} \rceil}} A_x(o_g^p) + \sum_{x \in \mathcal{C} \setminus \mathcal{C}_{\lceil \frac{d+1}{2} \rceil}} \max\{0, A_x(o_g^p)\}.$$

Case 5: Let the state of v be m_b^p . Here v is a bad vertex therefore, v 's children in S are overprotected. Let (v_1, v_2, \dots, v_d) be a descending ordering of \mathcal{C} according to values $\max\{A_{v_i}(o_g^p), A_{v_i}(o_b^p)\}$. We have the following recurrence relation:

$$A_v(m_b^p) = 1 + \sum_{x \in \mathcal{C}_{\lceil \frac{d+1}{2} \rceil - 1}} \max\{A_x(o_g^p), A_x(o_b^p)\}$$

For computation of $A_r(m_g^{\bar{p}})$ and $A_r(o_g^{\bar{p}})$, we replace d by $d - 1$ in the above recurrence relations as the root node r with d children has degree d , whereas any other non-leaf node with d children has degree $d + 1$.

The running time of this algorithm is easy to analyze. At each node $v \in V(T)$, we compute $A_v(s)$ where s is a state of v . The time required to get descending ordering of the children of v_i is $O(d_i \log d_i)$, where d_i is the number of children of vertex v_i . The number of subproblems is exactly the number of vertices in T . The total running time is therefore equal to $c \sum d_i \log d_i \leq c \log n \sum d_i = cn \log n = O(n \log n)$, where c is a constant.

5.2.1 An Illustration of the Algorithm

Consider the tree shown in Figure 5.1. For a leaf node $x \in \{x_2, x_4, x_6\}$, we have $A_x(0) = 0$,

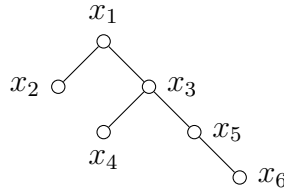


Figure 5.1: The largest locally minimal strong defensive alliances of T are $\{x_1, x_3, x_4\}$ and $\{x_1, x_3, x_5\}$

$A_x(m_g^p) = A_x(o_g^p) = A_x(o_g^{\bar{p}}) = A_x(m_g^{\bar{p}}) = A_x(o_b^p) = -\infty$ and $A_x(m_b^p) = 1$. For non-leaf node x_5 , we have $A_{x_5}(0) = \max\{A_{x_6}(0), A_{x_6}(m_g^{\bar{p}}), A_{x_6}(o_g^{\bar{p}})\} = \max\{0, -\infty, -\infty\} = 0$. Since $d = 1$, we have $A_{x_5}(m_g^p) = -\infty$ and $\lceil \frac{d+1}{2} \rceil - 1 = 0$. Therefore

$$A_{x_5}(m_g^{\bar{p}}) = 1 + \max\{A_{x_6}(m_g^p), A_{x_6}(m_b^p)\} = 1 + \max\{-\infty, 1\} = 2$$

and

$$A_{x_5}(o_g^p) = A_{x_5}(o_g^{\bar{p}}) = 1 + A_{x_5}(m_g^p) = 1 + (-\infty) = -\infty.$$

Next, we consider the states o_b^p and m_b^p . We have $A_{x_5}(o_b^p) = 1 + A_{x_5}(o_g^p) = 1 + (-\infty) = -\infty$ and $A_{x_5}(m_b^p) = 1$. For non-leaf node x_3 , we have $d = 2$;

$$\begin{aligned} A_{x_3}(0) &= \max \left\{ A_{x_4}(0), A_{x_4}(m_g^{\bar{p}}), A_{x_4}(o_g^{\bar{p}}), A_{x_5}(0), A_{x_5}(m_g^{\bar{p}}), A_{x_5}(o_g^{\bar{p}}) \right\} \\ &= \max\{0, -\infty, -\infty, 0, 2, -\infty\} = 2 \end{aligned}$$

and the associated locally minimal defensive alliance is $\{x_5, x_6\}$. For the state m_g^p , we have

$$\begin{aligned} A_{x_3}(m_g^p) &= 1 + \max_{x \in \mathcal{C}_1} \left\{ \max \left\{ A_x(1_{og}), A_x(1_{ob}) \right\} \right\} \\ &= 1 + \max \left\{ A_{x_4}(m_g^p), A_{x_4}(m_b^p), A_{x_5}(m_g^p), A_{x_5}(m_b^p) \right\} \\ &= 1 + \max\{-\infty, 1, -\infty, 1\} = 2. \end{aligned}$$

For the state $m_g^{\bar{p}}$, we have

$$\begin{aligned} A_{x_3}(m_g^{\bar{p}}) &= 1 + \max \left\{ \max \left\{ A_{x_4}(m_g^p), A_{x_4}(m_b^p) \right\} + \max \left\{ A_{x_5}(m_g^p), A_{x_5}(m_b^p), A_{x_5}(o_g^p), A_{x_5}(o_b^p) \right\}, \right. \\ &\quad \left. \max \left\{ A_{x_5}(m_g^p), A_{x_5}(m_b^p) \right\} + \max \left\{ A_{x_4}(m_g^p), A_{x_4}(m_b^p), A_{x_4}(o_g^p), A_{x_4}(o_b^p) \right\} \right\} \\ &= 1 + \max\{2, 2\} = 3, \end{aligned}$$

and the associated locally minimal defensive alliances are $\{x_3, x_4, x_5\}$ and $\{x_3, x_5, x_6\}$. Next, we get

$$\begin{aligned} A_{x_3}(o_g^p) &= 1 + \max \left\{ A_{x_4}(m_g^p) + \max \left\{ A_{x_5}(m_g^p), A_{x_5}(o_g^p), \right. \right. \\ &\quad \left. \left. A_{x_5}(m_b^p) + \max \left\{ A_{x_4}(m_g^p), A_{x_4}(o_g^p) \right\} \right\} \right\} \\ &= 1 + \max\{-\infty, -\infty\} = -\infty \end{aligned}$$

Similarly, since $A_{x_4}(m_g^p) = A_{x_5}(m_g^p) = -\infty$, we get $A_{x_3}(o_g^{\bar{p}}) = -\infty$. As $A_{x_4}(o_g^p) = A_{x_5}(o_g^p) = -\infty$, we get $A_{x_3}(o_b^p) = 1 + (-\infty) = -\infty$. As $A_{x_4}(o_g^p) = A_{x_4}(o_b^p) = A_{x_5}(o_g^p) = A_{x_5}(o_b^p) = -\infty$, we get $A_{x_3}(m_b^p) = 1 + (-\infty) = -\infty$. Finally, for the root node x_1 , we have $d = 2$ and

$$\begin{aligned} A_{x_1}(0) &= \max \left\{ A_{x_2}(0), A_{x_2}(m_g^{\bar{p}}), A_{x_2}(o_g^{\bar{p}}), A_{x_3}(0), A_{x_3}(m_g^{\bar{p}}), A_{x_3}(o_g^{\bar{p}}) \right\} \\ &= \max\{0, -\infty, -\infty, 2, 3, -\infty\} = 3. \end{aligned}$$

As $A_{x_2}(m_g^p) = A_{x_3}(m_g^p) = -\infty$, we get $A_{x_1}(o_g^{\bar{p}}) = -\infty$. Next

$$\begin{aligned} A_{x_1}(m_g^{\bar{p}}) &= 1 + \max \left\{ \max\{A_{x_2}(m_g^p), A_{x_2}(m_b^p)\}, \max\{A_{x_3}(m_g^p), A_{x_3}(m_b^p)\} \right\} \\ &= 1 + 2 = 3. \end{aligned}$$

The associated connected locally minimal strong defensive alliances are $\{x_1, x_3, x_4\}$ and $\{x_1, x_3, x_5\}$. Therefore, the size of the largest locally minimal strong defensive alliance is $\max\{A_{x_1}(0), A_{x_1}(m_g^{\bar{p}}), A_{x_1}(o_g^{\bar{p}})\} = 3$.

5.3 Locally Minimal Defensive Alliance in Planar Graphs is NP-complete

Bazgan, Fernau and Tuza showed in [1] that the LOCALLY MINIMAL STRONG DEFENSIVE ALLIANCE is NP-complete even when restricted to bipartite graphs with average degree less than 3.6. They proved that LOCALLY MINIMAL DEFENSIVE ALLIANCE is NP-complete even when restricted to bipartite graphs with average degree less than 5.6; and CONNECTED LOCALLY MINIMAL STRONG DEFENSIVE ALLIANCE is NP-complete even for bipartite graphs with average degree less than $2 + \epsilon$, for any $\epsilon > 0$, are NP-complete. Here we prove that the LOCALLY MINIMAL DEFENSIVE ALLIANCE problem is NP-complete in planar graphs, via a reduction from MINIMUM MAXIMAL MATCHING in cubic planar graph. Yannakakis and Gavril showed in [61] that the problem of finding a maximal matching of minimum size, is NP-hard in planar graphs of maximum degree 3. In [36], Horton and Kilakos obtained the NP-hardness of MINIMUM MAXIMAL MATCHING in planar cubic graphs.

Theorem 5.3.1. The LOCALLY MINIMAL DEFENSIVE ALLIANCE problem is NP-complete, even when restricted to planar graphs.

Proof. Clearly, the decision version of the problem belongs to NP. In order to obtain the NP-hardness result for the locally minimal defensive alliance problem, we obtain a polynomial reduction from the MINIMUM MAXIMAL MATCHING problem on cubic planar graphs proved NP-hard in [36]. Given an instance $I = (G, k)$ of the MINIMUM MAXIMAL MATCHING problem where G is a cubic planar graph, we construct an instance $I' = (G', k')$ of the LOCALLY MINIMAL DEFENSIVE ALLIANCE problem where G' is planar. An example is given

in Figure 5.2. The graph G' that we construct has vertex sets A and B , where $A = V(G) =$

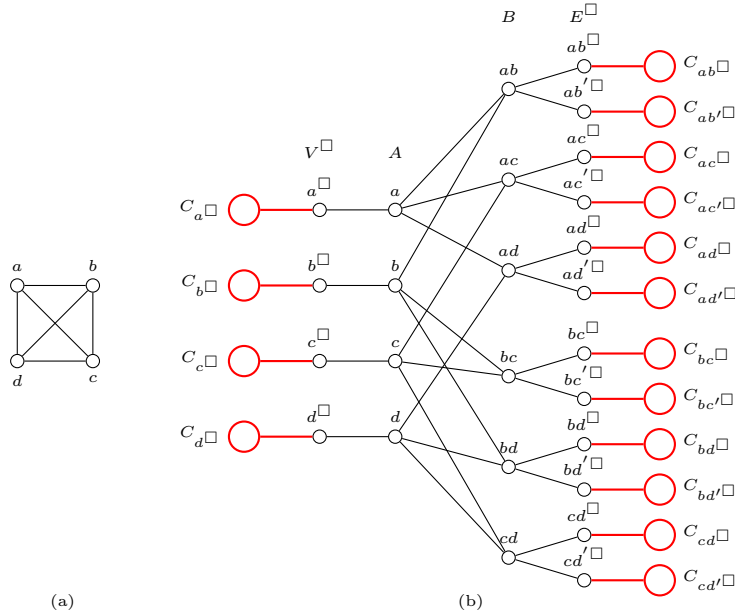


Figure 5.2: Reducing MINIMUM MAXIMAL MATCHING on planar cubic graphs to LOCALLY MINIMAL DEFENSIVE ALLIANCE on planar graphs. (a) An undirected graph $G = (V, E)$ with minimum maximal matching $M = \{(a, b), (c, d)\}$. (b) The planar graph G' produced by the reduction algorithm that has locally minimal defensive alliance $D = A \cup B \setminus \{ab, cd\} \cup \bigcup_{x \in V^\square \cup E^\square} \{x_1, x_2, x_4, x_5, x_7, x_8, \dots, x_{58}, x_{59}\}$. A red circle represents a cycle of length 60 and a red line between x and C_x indicates that x is adjacent to every vertex of C_x .

$\{v_1, v_2, \dots, v_n\}$ and $B = E(G) = \{e_1, e_2, \dots, e_m\}$, the edge set of G . We make v_i adjacent to e_j if and only if v_i is an endpoint of e_j . Further we add two sets of vertices $V^\square = \{v_1^\square, \dots, v_n^\square\}$ and $E^\square = \{e_1^\square, e_1'^\square, \dots, e_m^\square, e_m'^\square\}$; and make v_i adjacent to v_i^\square and e_i adjacent to e_i^\square and $e_i'^\square$. For each $x \in V^\square \cup E^\square$, we add a cycle C_x of length $6(n+m)$ with vertices $x_1, x_2, \dots, x_{6(n+m)}$ and make x adjacent to every vertex of C_x . This completes the construction of G' . It is easy to note that G' is a planar graph. Set $k' = 4(n+m)(n+2m) + (n+m-k)$. To complete the proof, we show that G has a maximal matching of size at most k if and only if G' has a locally minimal defensive alliance of size at least k' .

Suppose G has a maximal matching M of size at most k . We claim that $D = A \cup (B \setminus M) \cup \bigcup_{x \in V^\square \cup E^\square} \{x_1, x_2, x_4, x_5, x_7, x_8, \dots, x_{6(n+m)-2}, x_{6(n+m)-1}\}$ is a locally minimal defensive

alliance of size at least k' . It is easy to verify that all the vertices in A are protected. Clearly, all the vertices in

$$(B \setminus M) \cup \bigcup_{x \in V^\square \cup E^\square} \{x_1, x_2, x_4, x_5, x_7, x_8, \dots, x_{6(n+m)-2}, x_{6(n+m)-1}\}$$

are marginally protected. Since M is maximal, it is not possible to remove a vertex from $D \cap B$ since otherwise some vertex from A will become unprotected. Also, it is not possible to remove a vertex from $D \cap A$ since otherwise some vertices from $D \cap B$ will become unprotected. This shows that D is indeed a locally minimal defensive alliance.

For the reverse direction, suppose that G' has a locally minimal defensive alliance D of size at least k' . We claim that if $x \in V^\square \cup E^\square$, then x does not lie in D . For the sake of contradiction assume that $x \in D$. We consider two cases: *Case 1*: x is marginally protected. In this case, the cycle C_x contributes at most $3(n+m)$ vertices in D , as otherwise x is not marginally protected. Then $|D| < k'$ and this is a contradiction. *Case 2*: x is overprotected. Let x_i be a neighbour of x in $C_x \cap D$. As D is a locally minimal defensive alliance, x_i must have a marginally protected neighbour, say x_{i+1} , in C_x . Clearly, x_i and x_{i+1} are not marginally protected. This implies that x_i and x_{i+1} do not have a marginally protected neighbour, a contradiction to the assumption that D is a locally minimal defensive alliance. As $x \in V^\square \cup E^\square$ is not in D , then C_x can contribute at most $4(n+m)$ vertices in D by choosing two sets from the following three sets of vertices $X_0 = \{x_{3i} \mid 1 \leq i \leq 2(n+m)\}$, $X_1 = \{x_{3i+1} \mid 0 \leq i \leq 2(n+m) - 1\}$ and $X_2 = \{x_{3i+2} \mid 0 \leq i \leq 2(n+m) - 1\}$. We define $M = B \cap D^c$ and claim that there exists a set $M' \subseteq M$ such that M' is a maximal matching. If no subsets of M forms a maximal matching then clearly there exists an edge $(u, v) \in E$ such that it can still be added in M . It implies that all edges incident to u and v are in D and hence u and v are overprotected. Therefore, vertex $e = uv \in B$ does not have a marginally protected neighbour as both of its neighbors u and v are overprotected. This shows that there must exist a set $M' \subseteq M$ such that M' is a maximal matching and $|M'| \leq |M| \leq k$.

5.4 A color coding algorithm for EXACT CONNECTED LOCALLY MINIMAL DEFENSIVE ALLIANCE

In this section, we give a randomized FPT algorithm for the EXACT CONNECTED LOCALLY MINIMAL DEFENSIVE ALLIANCE problem using color coding technique. Let $G = (V, E)$ be a graph and let $S \subseteq V$ be a subset of size k . Every vertex in G is colored independently with one colour from the set $\{\text{green}, \text{red}\}$ with uniform probability. Denote the obtained coloring by $\chi : V(G) \rightarrow \{\text{red}, \text{green}\}$. A connected locally minimal defensive alliance S in G is called a green connected locally minimal defensive alliance in G with coloring χ if all the vertices in S are colored with green color and all the vertices in $N(S)$ are colored red.

Lemma 7. *Let G be a graph and let $\chi : V(G) \rightarrow \{\text{red}, \text{green}\}$ be a colouring of its vertices with two colours, chosen uniformly at random. Let $S \subseteq V$ be a connected locally minimal defensive alliance of size k in G . Then the probability that the elements of S are coloured with green colour and elements of $N(S)$ are coloured with red colour is at least $\frac{1}{2^{k^2+k}}$.*

Proof. As S is a (connected locally minimal) defensive alliance of size k in G , each element v in S is protected and therefore v can have at most k neighbours outside S . It follows that $|N(S)| \leq |S|k = k^2$. There are 2^n possible colorings χ ; and there are 2^{n-k^2-k} possible colorings where the k vertices of S are coloured green and at most k^2 neighbours of S are colored red. Hence the lemma follows. \square

Lemma 8. *Let G be a graph and let $\chi : V(G) \rightarrow \{\text{red}, \text{green}\}$ be a colouring of its vertices with two colours. Then there exists an algorithm that checks in time $O(n + m)$ whether G contains a green connected locally minimal defensive alliance of size k and, if this is the case, returns one such alliance.*

Proof. Let V_g and V_r be a partitioning of $V(G)$ such that all vertices in V_g are coloured green and all vertices in V_r are coloured red. A connected component C is said to be a green connected component if the vertices of C are colored green. Run DFS to identify all green connected components C_1, C_2, \dots, C_ℓ of $G[V_g]$ in $O(m+n)$ time. Then verify in linear time if there exists a green connected component C_i of size k that forms a locally minimal defensive alliance in G . \square

We now combine Lemma 7 and Lemma 8 to obtain the main result of this section.

Theorem 5.4.1. *There exists a randomized algorithm that, given an EXACT CONNECTED LOCALLY MINIMAL DEFENSIVE ALLIANCE instance (G, k) , in time $2^{\mathcal{O}(k^2+k)}(n + m)$ either reports a failure or finds a connected locally minimal defensive alliance of size exactly k in G . Moreover, if the algorithm is given a yes-instance, it returns a solution with a constant probability.*

Proof. Given an input instance (G, k) , we uniformly at random color the vertices of $V(G)$ with two colors green and red. That is, every vertex is colored independently with either green or red color with uniform probability. Denote the obtained coloring by $\chi : V(G) \rightarrow \{\text{red, green}\}$. We run the algorithm of Lemma 8 on the graph G with coloring χ . If it returns a green connected locally minimal defensive alliance S of size k , then we return this S as connected locally minimal defensive alliance of size k in G . Otherwise, we report failure.

It remains to bound the probability of finding a connected locally minimal defensive alliance of size k in the case (G, k) is a yes-instance. To this end, suppose G has a connected locally minimal defensive alliance S of size k . By Lemma 7, S becomes a green connected locally minimal defensive alliance of size k in the colouring χ with probability at least $\frac{1}{2^{k^2+k}}$. If this is the case, the algorithm of Lemma 8 finds a green connected locally minimal defensive alliance of size k (not necessarily S itself), and the algorithm returns a connected locally minimal defensive alliance of size k in G .

Thus we have an algorithm that runs in time $O(m + n)$ and given a yes-instance, returns a solution with probability at least $\frac{1}{2^{k^2+k}}$. Clearly, by repeating the algorithm independently 2^{k^2+k} times, we obtain the running time bound and the success probability at least $1 - \frac{1}{e}$. \square

5.5 FPT algorithm parameterized by neighbourhood diversity

In this section, we present an FPT algorithm for LOCALLY MINIMAL DEFENSIVE ALLIANCE problem parameterized by neighbourhood diversity. See Section 1.7 for the definitions of neighbourhood diversity and type graph. In this section, we prove the following theorem:

Theorem 5.5.1. *The LOCALLY MINIMAL DEFENSIVE ALLIANCE problem is fixed-parameter tractable when parameterized by the neighbourhood diversity.*

Let G be a connected graph such that $\text{nd}(G) = k$. Let C_1, \dots, C_k be the partition of $V(G)$ into sets of type classes. We assume $k \geq 2$ since otherwise the problem becomes trivial. Let H be the type graph of G . Next we guess the cardinality of $C_i \cap S$ and whether the vertices in C_i are marginally or overprotected, where S is a locally minimal defensive alliance. We make the following guesses:

- *Option 1:* $|C_i \cap S| = 0$.
- *Option 2:* $|C_i \cap S| = 1$ and the vertices in C_i are marginally protected.
- *Option 3:* $|C_i \cap S| = 1$ and the vertices in C_i are overprotected.
- *Option 4:* $|C_i \cap S| > 1$ and the vertices in C_i are marginally protected.
- *Option 5:* $|C_i \cap S| > 1$ and the vertices in C_i are overprotected.

There are at most 5^k choices for the tuple (C_1, C_2, \dots, C_k) as each C_i has 5 options as given above. Finally we reduce the problem of finding a locally minimal defensive alliance of maximum size to an integer linear programming optimization with k variables. Since integer linear programming is fixed parameter tractable when parameterized by the number of variables [47], we conclude that our problem is FPT when parameterized by the neighbourhood diversity.

ILP Formulation: Given a particular choice P of options for (C_1, C_2, \dots, C_k) , our goal here is to find a locally minimal defensive alliance of maximum size. For each C_i , we associate a variable x_i that indicates $|S \cap C_i| = x_i$. Clearly, $x_i = 0$, if C_i is assigned Option 1; $x_i = 1$ if C_i is assigned Option 2 or 3; and $x_i > 1$ if C_i is assigned Option 3 or 4. Because the vertices in C_i have the same neighbourhood, the variables x_i determine S uniquely, up to isomorphism. Let $\mathcal{C}_1 = \{C_i \mid x_i = 1\}$, $\mathcal{C}_{>1} = \{C_i \mid x_i > 1\}$ and $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_{>1}$. Let $H[\mathcal{C}]$ be the subgraph of H induced by \mathcal{C} . Now we label the vertices of $H[\mathcal{C}]$ as follows: vertex C_i is labelled c_1 if it is a clique and Option 2 is assigned to C_i ; vertex C_i is labelled $c_{>1}$ if it is a clique and Option 4 is assigned to C_i ; vertex C_i is labelled ind if it is an independent set

and Option 2 or 4 is assigned to C_i ; vertex C_i is labelled **op** if it is a clique or an independent set, and Option 3 or 5 is assigned to C_i . To ensure local minimality of defensive alliance, the induced subgraph must satisfy the following conditions:

- Every vertex labelled **op** in the induced graph must have at least one neighbour labelled $c_1, c_{>1}$ or **ind**.
- Every vertex labelled c_1 in the induced graph must have at least one neighbour labelled $c_1, c_{>1}$ or **ind**.
- Every vertex labelled **ind** in the induced graph must have at least one neighbour labelled $c_1, c_{>1}$ or **ind**.

Above conditions ensure local minimality of the solution because when we remove a vertex from the solution, we make sure at least one of its neighbours gets unprotected. This happens because every vertex in the solution has at least one neighbour which is marginally protected. If the induced subgraph $H[\mathcal{C}]$ satisfies all the above conditions then we proceed for the ILP, otherwise not. Let \mathcal{L} be a subset of \mathcal{C} consisting of all type classes which are cliques; $\mathcal{I} = \mathcal{C} \setminus \mathcal{L}$ be a subset of \mathcal{C} consisting of all type classes which are independent sets and $\mathcal{R} = \{C_1, \dots, C_k\} \setminus \mathcal{C}$. Let n_i denote the number of vertices in C_i . We consider two cases:

Case 1: Suppose $v \in C_j$ where $C_j \in \mathcal{I}$. Then the degree of v in S satisfies

$$d_S(v) = \sum_{C_i \in N_H(C_j) \cap \mathcal{C}} x_i \quad (5.1)$$

Thus, including itself, v has $1 + \sum_{C_i \in N_H(C_j) \cap \mathcal{C}} x_i$ defenders in G . Note that if $C_i \in \mathcal{C}$, then only x_i vertices of C_i are in S and the remaining $n_i - x_i$ vertices of C_i are outside S . The degree of v outside S satisfies

$$d_{S^c}(v) = \sum_{C_i \in N_H(C_j) \cap \mathcal{C}} (n_i - x_i) + \sum_{C_i \in N_H(C_j) \cap \mathcal{R}} n_i \quad (5.2)$$

Case 2: Suppose $v \in C_j$ where $C_j \in \mathcal{L}$. The degree of v in S satisfies

$$d_S(v) = \sum_{C_i \in N_H[C_j] \cap \mathcal{C}} x_i \quad (5.3)$$

The degree of v outside S satisfies

$$d_{S^c}(v) = \sum_{C_i \in N_H[C_j] \cap \mathcal{C}} (n_i - x_i) + \sum_{C_i \in N_H[C_j] \cap \mathcal{R}} n_i \quad (5.4)$$

In the following, we present an ILP formulation of locally minimal defensive alliance problem, where a choice of options for (C_1, \dots, C_k) is given:

$$\begin{aligned}
& \text{Maximize } \sum_{C_i \in \mathcal{C}} x_i \\
& \text{Subject to} \\
& 1 + \sum_{C_i \in N_H(C_j) \cap \mathcal{C}} 2x_i > \sum_{C_i \in N_H(C_j)} n_i, \text{ for all } C_j \in \mathcal{I}, \text{ labelled op,} \\
& \sum_{C_i \in N_H(C_j) \cap \mathcal{C}} 2x_i - \sum_{C_i \in N_H(C_j)} n_i = 0 \text{ or } -1, \text{ for all } C_j \in \mathcal{I}, \text{ labelled ind,} \\
& \sum_{C_i \in N_H[C_j] \cap \mathcal{C}} 2x_i > \sum_{C_i \in N_H[C_j]} n_i, \text{ for all } C_j \in \mathcal{L}, \text{ labelled op,} \\
& \sum_{C_i \in N_H[C_j] \cap \mathcal{C}} 2x_i - \sum_{C_i \in N_H[C_j]} n_i = 0 \text{ or } 1, \text{ for all } C_j \in \mathcal{L}, \text{ labelled } c_1 \text{ or } c_{>1}, \\
& x_i = 1 \text{ for all } i : C_i \in \mathcal{C}_1; \\
& x_i \in \{2, 3, \dots, |C_i|\} \text{ for all } i : C_i \in \mathcal{C}_{>1}.
\end{aligned}$$

Solving the ILP: In the formulation for LOCALLY MINIMAL DEFENSIVE ALLIANCE problem, we have at most k variables. The value of objective function is bounded by n and the value of any variable in the integer linear programming is also bounded by n . The constraints can be represented using $O(k^2 \log n)$ bits. Lemma 1 of Section 2.1.3 implies that we can solve the problem with the guess P in FPT time. There are at most 5^k choices for P , and the ILP formula for a guess can be solved in FPT time. Thus Theorem 6.3.1 holds.

5.6 Hardness of EXACT CONNECTED LOCALLY MINIMAL DEFENSIVE ALLIANCE parameterized by treewidth

In this section, we prove the following theorem:

Theorem 5.6.1. *The EXACT CONNECTED LOCALLY MINIMAL DEFENSIVE ALLIANCE problem is $W[1]$ -hard when parameterized by the treewidth of the input graph.*

We introduce several variants of LOCALLY MINIMAL DEFENSIVE ALLIANCE problem that we require in our proofs. The problem LOCALLY MINIMAL DEFENSIVE ALLIANCE^F generalizes LOCALLY MINIMAL DEFENSIVE ALLIANCE where some vertices are forced to be outside the solution (these vertices are called *forbidden* vertices). This variant can be formalized as follows:

LOCALLY MINIMAL DEFENSIVE ALLIANCE^F
Input: A graph $G = (V, E)$, an integer k and a set $V_{\square} \subseteq V(G)$.
Question: Does there exist a locally minimal defensive alliance $S \subseteq V(G)$ of size at least k such that $S \cap V_{\square} = \emptyset$?

LOCALLY MINIMAL DEFENSIVE ALLIANCE^{FN} is a further generalization where some vertices are forced to be in every solution (these vertices are called *necessary* vertices) and some other vertices are forced to be outside the solution. This variant can be formalized as follows:

LOCALLY MINIMAL DEFENSIVE ALLIANCE^{FN}
Input: A graph $G = (V, E)$, an integer k , a set $V_{\Delta} \subseteq V(G)$, and a set $V_{\square} \subseteq V(G)$.
Question: Does there exist a locally minimal defensive alliance $S \subseteq V(G)$ of size at least k such that (i) $V_{\Delta} \subseteq S$, (ii) $V_{\square} \subseteq S^c$?

Let $G = (V, E)$ be an undirected and edge weighted graph, where V , E , and w denote the set of nodes, the set of edges and a positive integral weight function $w : E \rightarrow Z^+$, respectively. An orientation Λ of G is an assignment of a direction to each edge $\{u, v\} \in E(G)$, that is, either (u, v) or (v, u) is contained in Λ . The weighted outdegree of u on Λ is $w_{\text{out}}^u = \sum_{(u,v) \in \Lambda} w(\{u, v\})$. We define MINIMUM MAXIMUM OUTDEGREE problem as follows:

MINIMUM MAXIMUM OUTDEGREE

Input: A graph G , an edge weighting w of G given in unary, and a positive integer r .

Question: Is there an orientation Λ of G such that $w_{\text{out}}^u \leq r$ for each $u \in V(G)$?

It is known that MINIMUM MAXIMUM OUTDEGREE is W[1]-hard when parameterized by the treewidth of the input graph [57]. To prove Theorem 6.5.1, we reduce MINIMUM MAXIMUM OUTDEGREE to LOCALLY MINIMAL DEFENSIVE ALLIANCE^{FN} and then we show how we can successively reduce the latter problem CONNECTED LOCALLY MINIMAL DEFENSIVE ALLIANCE^{FN}, CONNECTED LOCALLY MINIMAL DEFENSIVE ALLIANCE^F and finally EXACT CONNECTED LOCALLY MINIMAL DEFENSIVE ALLIANCE.

5.6.1 Hardness of EXACT CONNECTED LOCALLY MINIMAL DEFENSIVE ALLIANCE with Forbidden and Necessary Vertices

To show W[1]-hardness of LOCALLY MINIMAL DEFENSIVE ALLIANCE^{FN}, we reduce from MINIMUM MAXIMUM OUTDEGREE which is known to be W[1]-hard [57] parameterized by the treewidth of the graph:

Lemma 9. *The LOCALLY MINIMAL DEFENSIVE ALLIANCE^{FN} is W[1]-hard when parameterized by the treewidth of the input graph.*

Proof. Let $G = (V, E, w)$ and a positive integer r be an instance of MINIMUM MAXIMUM OUTDEGREE. We construct an instance of LOCALLY MINIMAL DEFENSIVE ALLIANCE^{FN} as follows. We illustrate our construction in Figure 5.3. Before we formally define our reduction, we briefly describe the intuition. We introduce three types of vertices: necessary vertices, forbidden vertices and normal vertices. We often indicate necessary vertices by means of a triangular node shape, and forbidden vertices by means of a square node shape, and normal vertices by means of a circular node shape. We want to make sure that *necessary vertices* are always inside every solution and *forbidden vertices* are always outside every solution; and a *normal vertex* could be inside or outside the solution. For each vertex $v \in V(G)$, we introduce one new forbidden vertex v^\square , one set of normal vertices $H_v = \{h_1^v, \dots, h_{2r}^v\}$ and one set of forbidden $H_v^\square = \{h_1^{v^\square}, \dots, h_{2r}^{v^\square}\}$. For each edge $(u, v) \in E(G)$, we introduce two sets of

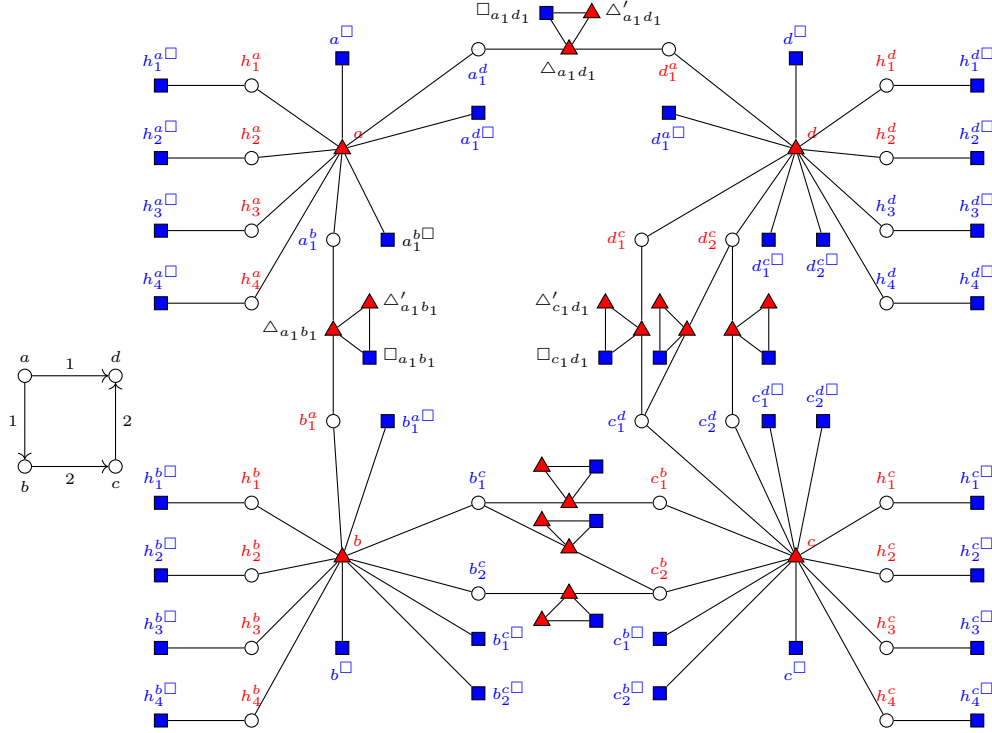


Figure 5.3: Result of our reduction on a MINIMUM MAXIMUM OUTDEGREE instance G with $r = 2$. The graph G long with its orientation is shown at the left; and G' is shown at the right. The necessary vertices in V_{Δ} are filled with red colour; the forbidden vertices in V_{\square} are filled with blue colour. The vertices in locally minimal defensive alliance S are shown in red label for the given orientation of G .

normal vertices $V_{uv} = \{u_1^v, \dots, u_{w(u,v)}^v\}$ and $V_{vu} = \{v_1^u, \dots, v_{w(u,v)}^u\}$; and two sets of forbidden vertices $V_{uv}^{\square} = \{u_1^{v\square}, \dots, u_{w(u,v)}^{v\square}\}$ and $V_{vu}^{\square} = \{v_1^{u\square}, \dots, v_{w(u,v)}^{u\square}\}$. Let $\omega = \sum_{(u,v) \in E(G)} w(u,v)$. We define a set of complementary vertex pairs

$$C = \left\{ (u_i^v, v_i^u), \mid (u,v) \in E(G), 1 \leq i \leq w(u,v) \right\} \\ \cup \left\{ (u_{i+1}^v, v_i^u) \mid (u,v) \in E(G), 1 \leq i \leq w(u,v) - 1 \right\}.$$

For every complementary pair $c = (u^v, v^u) \in C$, we introduce a set $V(c) = \{\Delta_{u^v v^u}, \square_{u^v v^u}, \Delta'_{u^v v^u}\}$ of three vertices: two necessary vertices and one forbidden vertex. We add the following set of edges to the complementary pair gadget

$$E(c) = \{(\Delta_{u^v v^u}, \square_{u^v v^u}), (\Delta_{u^v v^u}, \Delta'_{u^v v^u}), (\Delta'_{u^v v^u}, \square_{u^v v^u}), (u^v, \Delta_{u^v v^u}), (\Delta_{u^v v^u}, v^u)\}.$$

We now define the graph G' with

$$\begin{aligned} V(G') &= V(G) \cup \{v^\square \mid v \in V(G)\} \cup \bigcup_{v \in V(G)} (H_v \cup H_v^\square) \\ &\cup \bigcup_{(u,v) \in E(G)} (V_{uv} \cup V_{uv}^\square \cup V_{vu} \cup V_{vu}^\square) \cup \bigcup_{c \in C} V(c) \end{aligned}$$

and

$$\begin{aligned} E(G') &= \{(v, h) \mid v \in V(G), h \in H_v\} \cup \{(h_i^v, h_i^{v^\square}) \mid 1 \leq i \leq 2r, v \in V(G)\} \\ &\cup \{(u, x) \mid (u, v) \in E(G), x \in V_{uv} \cup V_{uv}^\square\} \\ &\cup \{(x, v) \mid (u, v) \in E(G), x \in V_{vu} \cup V_{vu}^\square\} \cup \{(v, v^\square) \mid v \in V(G)\} \\ &\cup \bigcup_{c \in C} E(c). \end{aligned}$$

The number of vertices in G' is $n(4r + 2) + 4\omega + 3|C|$, where n is the number of vertices in G . Finally we define the set of necessary vertices

$$V_\Delta = V(G) \cup \bigcup_{(u^v, v^u) \in C} \{\Delta_{u^v v^u}, \Delta'_{u^v v^u}\}$$

and the set of forbidden vertices

$$V_\square = \bigcup_{(u,v) \in E(G)} (V_{uv}^\square \cup V_{vu}^\square) \cup \bigcup_{v \in V(G)} (H_v^\square \cup \{v^\square\}) \cup \bigcup_{(u^v, v^u) \in C} \{\square_{u^v v^u}\}.$$

We set $k = n(r + 1) + \omega + 2|C|$. Note that the end points of each complementary pair are adjacent to a vertex in V_Δ . We use I to denote $(G', k, V_\Delta, V_\square)$ which is an instance of LOCALLY MINIMAL DEFENSIVE ALLIANCE^{FN}.

Clearly, it takes polynomial time to compute I . We now prove that the treewidth of the graph G' of I is bounded by a function of the treewidth of G . We do so by modifying an optimal tree decomposition τ of G as follows:

- For every edge (u, v) of G , there is a node in τ whose bag B contains both u and v ; add to this node a chain of nodes $1, 2, \dots, w(u, v) - 1$ where the bag of node i is

$$B \cup \{u_i^v, v_i^u, u_{i+1}^v, v_{i+1}^u, \Delta_{u_i^v v_i^u}, \square_{u_i^v v_i^u}, \Delta'_{u_i^v v_i^u}, \Delta_{u_i^v v_{i+1}^u}, \square_{u_i^v v_{i+1}^u}, \Delta'_{u_i^v v_{i+1}^u}\}.$$

- For every edge (u, v) of G , there is a node in τ whose bag B contains u ; add to this node a chain of nodes $1, 2, \dots, w(u, v)$ where the bag of node i is $B \cup \{u_i^{v\square}, u^\square\}$.
- For every edge (u, v) of G , there is a node in τ whose bag B contains v and add to this node a chain of nodes $1, 2, \dots, w(u, v)$ where the bag of node i is $B \cup \{v_i^{u\square}, v^\square\}$.
- For every vertex v of G , there is a node in τ whose bag B contains v and add to this node a chain of nodes $1, 2, \dots, 2r$ where the bag of node i is $B \cup \{h_i^v, h_i^{v\square}\}$.

Clearly, the modified tree decomposition is a valid tree decomposition of G' and its width is at most the treewidth of G plus ten.

Let D be the directed graph obtained by an orientation of the edges of G such that for each vertex the sum of the weights of outgoing edges is at most r . Let w_{out}^x and w_{in}^x denote the sum of the weights of outgoing and incoming edges of vertex x , respectively. Then the set

$$S = V_\Delta \cup \bigcup_{(u,v) \in E(D)} V_{vu} \cup \{h_i^v \mid 1 \leq i \leq r + w_{\text{out}}^v; v \in V(G)\}$$

is a locally minimal defensive alliance of size k . Note that $(u, v) \in E(D)$ is a directed edge from u to v in D . To prove that S is a locally minimal defensive alliance, we show that $d_S(x) \geq d_{S^c}(x)$ for all $x \in S$ and each vertex in S has a marginally protected neighbour. Let x be an arbitrary element of S . If $x \in H_v$, then including itself, it has two neighbour in S and one neighbour in S^c ; so x is marginally protected. If $x \in V_{uv}$, then including itself, it has three neighbour in S and no neighbours in S^c ; so x is protected. If $x \in V(G) \subseteq V_\Delta$, then including itself, it has $r + w_{\text{out}}^x + w_{\text{in}}^x + 1$ neighbours in S , and it has $r + w_{\text{out}}^x + w_{\text{in}}^x + 1$ neighbours in S^c ; so x is marginally protected. If $x \in V_\Delta \setminus V(G) = \bigcup_{(u^v, v^u) \in C} \{\Delta_{u^v v^u}, \Delta'_{u^v v^u}\}$, then it is easy to see that x is marginally protected. Thus every vertex x in S is protected and has at least one marginally protected neighbour.

Conversely, suppose S is a solution of the instance I . We first show that exactly one vertex from each complementary pair belongs to S . The following holds for every solution S of I : As S contains $\Delta_{u^v v^u}, \Delta'_{u^v v^u}$ for every complementary pair $(u^v, v^u) \in C$, it must include at least one vertex from $\{u^v, v^u\}$ to protect $\Delta_{u^v v^u}$ in the solution. We claim S cannot include both u^v and v^u . We know every vertex in a locally minimal defensive alliance must have a

marginally protected neighbour. As $\Delta_{u^v v^u}$ is the only neighbour of $\Delta'_{u^v v^u}$ in S , $\Delta_{u^v v^u}$ must be marginally protected. Suppose both u^v and v^u are in S , then including itself, $\Delta_{u^v v^u}$ has four neighbours in S , and one neighbour outside S . Then $\Delta_{u^v v^u}$ is not marginally protected. Therefore S contains exactly one endpoint for each $(u^v, v^u) \in C$. For every $(u, v) \in E(G)$, either $V_{uv} \in S$ or $V_{vu} \in S$ due to the complementary vertex pairs. We define a directed graph D by $V(D) = V(G)$ and

$$E(D) = \left\{ (u, v) \mid V_{vu} \subseteq S \right\} \cup \left\{ (v, u) \mid V_{uv} \subseteq S \right\}.$$

Suppose there is a vertex x in D for which $w_{\text{out}}^x > r$. Clearly $x \in V(G)$. We know, including itself, x has at most $2r + w_{\text{in}}^x + 1$ neighbours in S and x has at least $2w_{\text{out}}^x + w_{\text{in}}^x + 1$ neighbours in S^c . Then $d_{S^c}(x) > d_S(x)$, as by assumption $w_{\text{out}}^x > r$, a contradiction to the fact that S is a (locally minimal) defensive alliance of G' . Hence $w_{\text{out}}^x \leq r$ for all $x \in V(D)$. \square

Lemma 10. *The CONNECTED LOCALLY MINIMAL DEFENSIVE ALLIANCE^{FN} is $W[1]$ -hard when parameterized by the treewidth of the graph.*

Proof. Let $I = (G, k, V_{\Delta}, V_{\square})$ be a LOCALLY MINIMAL DEFENSIVE ALLIANCE^{FN} instance; let n denote the number of vertices in G . We construct an instance $I' = (G', k', V'_{\Delta}, V'_{\square})$ of CONNECTED LOCALLY MINIMAL DEFENSIVE ALLIANCE^{FN} the following way. We illustrate our construction in Figure 5.4. The construction of G' starts with $G' := G$ and then add the following new vertices and edges. We introduce one necessary vertex h and one forbidden vertex t , and make them adjacent to every vertex of G . We introduce a set of necessary vertices $H = \{h_1, h_2, \dots, h_{4n}\}$. For every vertex h_i in H , we introduce four forbidden vertices $H_i^{\square} = \{h_i^{1\square}, h_i^{2\square}, h_i^{3\square}, h_i^{4\square}\}$. We now define the CONNECTED LOCALLY MINIMAL DEFENSIVE ALLIANCE^{FN} instance $I' = (G', k', V'_{\Delta}, V'_{\square})$ where

$$V'_{\Delta} = V_{\Delta} \cup \{h\} \cup \bigcup_{i=1}^{4n} \{h_i\},$$

$$V'_{\square} = V_{\square} \cup \{t\} \cup \bigcup_{i=1}^{4n} \bigcup_{j=1}^4 \{h_i^{j\square}\}$$

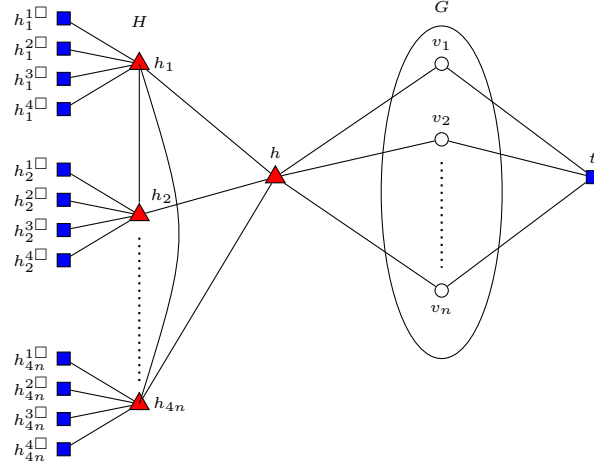


Figure 5.4: The graph G' in our reduction from LOCALLY MINIMAL DEFENSIVE ALLIANCE^{FN} to CONNECTED LOCALLY MINIMAL DEFENSIVE ALLIANCE^{FN} in the proof of Lemma 10.

and G' is the graph defined by

$$V(G') = V(G) \cup \{h, t\} \cup \bigcup_{i=1}^{4n} \bigcup_{j=1}^4 \{h_i, h_i^{j\Box}\}$$

and

$$E(G') = E(G) \cup \bigcup_{u \in V(G)} \{(u, h), (u, t)\} \cup \bigcup_{i=1}^{4n} \bigcup_{j=1}^4 \{(h_i, h_i^{j\Box}), (h, h_i)\} \cup \bigcup_{i=1}^{4n-1} \{(h_i, h_{i+1}), (h_{4n}, h_1)\}$$

We set $k' = k + 4n + 1$. The treewidth of G' is equal to the treewidth of G plus 5. We now claim that I is a yes-instance if and only if I' is a yes-instance. Suppose there exists a locally minimal defensive alliance $S \subseteq V(G)$ with $|S| \geq k$ of G such that $V_\Delta \subseteq S$ and $V_\square \cap S = \emptyset$. We claim that the set

$$S' = S \cup H \cup \{h\}$$

is a connected locally minimal defensive alliance of G' such that $|S'| \leq k'$, $V'_\Delta \subseteq S'$ and $V'_\square \cap S' = \emptyset$. As h is adjacent to all vertices in S' , the subgraph induced by S' is connected. In G' , each vertex of $V(G)$ is adjacent to one new necessary vertex h and one new forbidden vertex t . Therefore each vertex in S remains protected and has a marginally protected neighbour. Each vertex in H has four neighbours inside S' and four neighbours outside S' .

Thus each vertex in H is marginally protected and has a marginally protected neighbour in H . Similarly, h is strongly protected and adjacent to marginally protected vertices in H . Since we are adding the set H of $4n$ vertices and a vertex h to S , it implies that $|S'| \geq k'$. This shows that I' is a yes-instance.

Conversely, suppose that there exists a connected locally minimal defensive alliance S' of G' with $|S'| \geq k'$ such that $V'_\Delta \subseteq S'$ and $V'_\square \cap S' = \emptyset$. We claim that $S = S' \cap V(G)$ forms a locally minimal defensive alliance in G . Clearly, each vertex in $S = S' \cap V(G)$ remains protected as it loses one neighbour from inside the solution and one neighbour from outside the solution. This also implies that marginally protected vertices in S' will remain marginally protected in S . As every vertex in S' had a marginally protected neighbour in S' and h is strongly protected in S' , it shows that every vertex in S will also have a marginally protected neighbour in S . As we are removing exactly $4n + 1$ vertices from the solution, we get $|S| \geq k$. This shows that I is a yes-instance. \square

As a consequence of Lemma 10, we have the following result:

Corollary 5. The EXACT CONNECTED LOCALLY MINIMAL DEFENSIVE ALLIANCE^{FN} problem is W[1]-hard when parameterized by the treewidth of the graph.

5.6.2 Hardness of CONNECTED LOCALLY MINIMAL DEFENSIVE ALLIANCE with Forbidden Vertices

Now we present a transformation that eliminates necessary vertices. The basic idea is that we ensure that a necessary vertex u is always part of every solution by adding a very large gadget which is connected to G through u only. If u is not in the solution, then we cannot include any vertex from the newly added gadget to the connected locally minimal defensive alliance; as a result we will not be able to attain the required size of the solution. Therefore, we are forced to include every necessary vertex u in the solution.

Lemma 11. *The CONNECTED LOCALLY MINIMAL DEFENSIVE ALLIANCE^F is W[1]-hard when parameterized by the treewidth of the graph.*

Proof. Let $I = (G, k, V_\Delta, V_\square)$ be a CONNECTED LOCALLY MINIMAL DEFENSIVE ALLIANCE^{FN}

instance. We construct an instance $I' = (G', k', V'_\square)$ of CONNECTED LOCALLY MINIMAL DEFENSIVE ALLIANCE^F the following way. We illustrate our construction in Figure 5.5. The

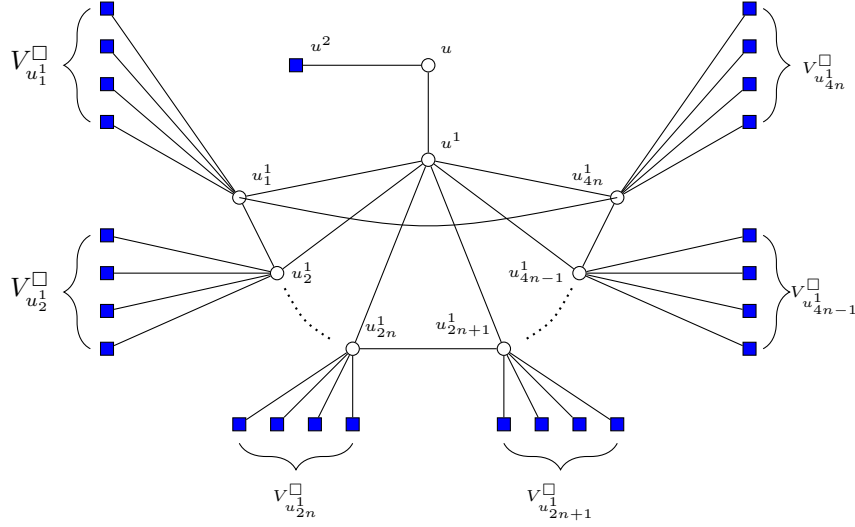


Figure 5.5: An illustration of the reduction of necessary vertices in CONNECTED LOCALLY MINIMAL DEFENSIVE ALLIANCE^{FN} to CONNECTED LOCALLY MINIMAL DEFENSIVE ALLIANCE^F. The vertex u may have additional neighbours in G .

construction of G' starts with $G' := G$ and then add the following new vertices and edges. For each $u \in V_\Delta$, we introduce two new vertices u^1 and u^2 . For each u^1 , we introduce a set $V_{u^1} = \{u_i^1 \mid 1 \leq i \leq 4n\}$ of $4n$ vertices. For each $u_i^1 \in V_{u^1}$, we add a set $V_{u_i^1}^\square$ of four one-degree forbidden vertices and make them adjacent to u_i^1 . We now define the graph G' with

$$V(G') = V(G) \cup \bigcup_{u \in V_\Delta} \{u^1, u^2\} \cup \bigcup_{u \in V_\Delta} \bigcup_{i=1}^{4n} \{u_i^1\} \cup \bigcup_{u \in V_\Delta} \bigcup_{i=1}^{4n} V_{u_i^1}^\square$$

and

$$E(G') = E(G) \cup \bigcup_{u \in V_\Delta} \bigcup_{i=1}^{4n} \left\{ (u^1, u), (u^2, u), (u^1, u_i^1), (u_i^1, \alpha) \mid \alpha \in V_{u_i^1}^\square \right\}.$$

We set

$$V'_\square = V_\square \cup \bigcup_{u \in V_\Delta} \{u^2\} \cup \bigcup_{u \in V_\Delta} \bigcup_{i=1}^{4n} V_{u_i^1}^\square$$

and $k' = k + (4n + 1)|V_\Delta|$.

We claim that I is a yes-instance if and only if I' is a yes-instance. Suppose there is a connected locally minimal defensive alliance S of size at least k in G such that $V_\Delta \subseteq S$ and $V_\square \cap S = \emptyset$. We claim the set

$$S' = S \cup \bigcup_{u \in V_\Delta} \bigcup_{i=1}^{4n} \{u^1, u_i^1\}$$

is a connected locally minimal defensive alliance of size at least k' in G' such that $S' \cap V'_\square = \emptyset$. Clearly, each vertex in S remains protected and has a marginally protected neighbour because we are adding equal number of neighbours inside and outside the solution for each vertex in S . We observe that the vertices in $V_{u^1} = \{u_i^1 \mid 1 \leq i \leq 4n\}$ are marginally protected and have a marginally protected neighbour in the same set. Also, u^1 is overprotected and has marginally protected neighbours in the set V_{u^1} . As for every vertex in the set V_Δ , we are adding exactly $4n+1$ vertices to the solution, it implies that $|S'| \geq k' = k + (4n+1)|V_\Delta|$. This proves that if I is a yes-instance then I' is a yes-instance.

To prove the reverse direction of the equivalence, suppose there is a connected locally minimal defensive alliance S' of size at least k' in G' such that $V'_\square \cap S' = \emptyset$. We first prove that all the vertices $\{u, u^1 \mid u \in V_\Delta\}$ are in the solution. If there is a vertex u^1 such that it is not part of S' then no vertices from the set V_{u^1} are part of S' as they will not be protected; every vertex of V_{u^1} will have at least five neighbours outside S' and at most three neighbours, including itself, inside S' . This implies that $|S'| \leq n + (4n+1)(|V_\Delta| - 1) < (4n+1)|V_\Delta| < k'$, a contradiction to the fact that $|S'| \geq k'$. If there is a vertex u such that it is not part of S' then no vertices from the set V_{u^1} are part of S' as the solution will be disconnected; if V_{u^1} itself forms a solution then clearly $|V_{u^1}| < k'$. We claim that $S = S' \cap V(G)$ is a locally minimal defensive alliance in graph G of size at least k such that $V_\Delta \subseteq S$. We proved that $V_\Delta \subseteq S$. Since every vertex in S loses equal number of neighbours from inside and outside the solution, the vertices of S remain protected. This also implies that every vertex in S has a marginally protected neighbour. This is true since u has only one new neighbour u^1 in S' and since u^1 is overprotected in S' , u must have a marginally protected neighbour in $S = S' \cap V(G)$. As we are removing exactly $(4n+1)|V_\Delta|$ vertices from S' , we get $|S| \geq k$. \square

As a consequence of this lemma, we have the following result:

Corollary 6. The EXACT CONNECTED LOCALLY MINIMAL DEFENSIVE ALLIANCE^F prob-

lem is $W[1]$ -hard when parameterized by the treewidth of the input graph.

5.6.3 Hardness of EXACT CONNECTED LOCALLY MINIMAL DEFENSIVE ALLIANCE

We now introduce a transformation that eliminates forbidden vertices. The basic idea is that we ensure that a forbidden vertex u is never part of a solution by adding a large number of neighbors to u such that we could only defend u by exceeding the solution size.

Lemma 12. *The EXACT CONNECTED LOCALLY MINIMAL DEFENSIVE ALLIANCE problem is $W[1]$ -hard when parameterized by the treewidth of the graph.*

Proof. Let $I = (G, k, V_\square)$ be an instance of the EXACT CONNECTED LOCALLY MINIMAL DEFENSIVE ALLIANCE^F problem. We construct an instance $I' = (G', k')$ of EXACT CONNECTED LOCALLY MINIMAL DEFENSIVE ALLIANCE as follows. The construction of G' starts with $G' := G$ and then add the following new vertices and edges. For every $u \in V_\square$, we introduce a set V_u of $2n + 2$ one degree vertices adjacent to u . The graph G' is defined as follows:

$$V(G') = V(G) \cup \bigcup_{u \in V_\square} V_u$$

and

$$E(G') = E(G) \cup \bigcup_{u \in V_\square} \{(u, \beta) \mid \beta \in V_u\}$$

We set $k' = k (\leq n)$.

We claim that I is a yes-instance if and only if I' is a yes-instance. Suppose S is a connected locally minimal defensive alliance of size k in G such that $S \cap V_\square = \emptyset$. Then S is also a connected locally minimal defensive alliance of size k in G' . Conversely, suppose that S' is a connected locally minimal defensive alliance of size $k (\leq n)$ in G' . We observe that S' cannot contain any vertex from the set V_\square as the protection of a vertex from V_\square requires at least $n + 1$ vertices in S' . This implies that $|S'| \geq n + 1$, a contradiction to the fact that $|S'| = k \leq n$. Therefore, S' is an exact connected locally minimal defensive alliance of size k in G such that $V_\square \cap S' = \emptyset$. This shows that I is a yes-instance.

This proves Theorem 6.5.1.

5.7 Graphs of bounded treewidth

In this section we prove that LOCALLY MINIMAL DEFENSIVE ALLIANCE problem can be solved in polynomial time for graphs of bounded treewidth. In other words, this section presents an XP-algorithm for LOCALLY MINIMAL DEFENSIVE ALLIANCE problem parameterized by treewidth. We now prove the following theorem:

Theorem 5.7.1. Given an n -vertex graph G and its nice tree decomposition T of width at most k , the size of a maximum locally minimal defensive alliance of G can be computed in $8^k n^{O(2^{k+1})}$ time.

Let $(T, \{X_t\}_{t \in V(T)})$ be a nice tree decomposition rooted at node r of the input graph G . For a node t of T , let V_t be the union of all bags present in the subtree of T rooted at t , including X_t . We denote by G_t the subgraph of G induced by V_t . For each node t of T , we construct a table $dp_t(A, \mathbf{x}, a, \alpha, \mathbf{y}, \mathbf{z}, \beta) \in \{\text{true}, \text{false}\}$ where $A \subseteq X_t$; \mathbf{x} and \mathbf{y} are vectors of length n ; a , α and β are integers between 0 and n . We set $dp_t(A, \mathbf{x}, a, \alpha, \mathbf{y}, \mathbf{z}, \beta) = \text{true}$ if and only if there exists a set $A_t \subseteq V_t$ such that:

1. $A_t \cap X_t = A$
2. $a = |A_t|$
3. the i th coordinate of vector \mathbf{x} is

$$x(i) = \begin{cases} d_{A_t}(v_i) & \text{for } v_i \in A \\ 0 & \text{otherwise} \end{cases}$$

4. α is the number of vertices $v \in A_t$ that are protected, that is, $d_{A_t}(v) \geq \frac{d_G(v)-1}{2}$.
5. A vertex $v \in A$ is said to be "good" if it has at least one marginally protected neighbour in $A_t \setminus A$. A vertex $v \in A$ is said to be "bad" if it has no marginally protected neighbours

in $A_t \setminus A$. Here \mathbf{y} is a vector of length n , and the i th coordinate of vector \mathbf{y} is

$$y(i) = \begin{cases} g & \text{if } v_i \in A \text{ and } v_i \text{ is a good vertex} \\ b & \text{if } v_i \in A \text{ and } v_i \text{ is a bad vertex} \\ 0 & \text{otherwise} \end{cases}$$

6. \mathbf{z} is a 2^k length vector, where the entry $z(S)$ associated with subset $S \subseteq A$ denotes the number of common bad neighbours of S in $A_t \setminus A$. The z vector considers the power set of A in lexicographic order. For example, let $A = \{a, b, c\}$, then $\mathbf{z} = \left(z(\{a\}), z(\{a, b\}), z(\{a, b, c\}), z(\{a, c\}), z(\{b\}), z(\{b, c\}), z(\{c\}) \right)$.
7. β is the number of good vertices in A_t .

We compute all entries $dp_t(A, \mathbf{x}, a, \alpha, \mathbf{y}, \mathbf{z}, \beta)$ in a bottom-up manner. Since $tw(T) \leq k$, at most $2^k n^k (n+1)^3 2^k n^{2^k} = 4^k n^{O(2^k)}$ records are maintained at each node t . Thus, to prove Theorem 5.7.1, it suffices to show that each entry $dp_t(A, \mathbf{x}, a, \alpha, \mathbf{y}, \mathbf{z}, \beta)$ can be computed in $2^k n^{O(2^k)}$ time, assuming that the entries for the children of t are already computed.

Leaf node: For a leaf node t we have that $X_t = \emptyset$. Thus $dp_t(A, \mathbf{x}, a, \alpha, \mathbf{y}, \mathbf{z}, \beta)$ is true if and only if $A = \emptyset$, $\mathbf{x} = \mathbf{0}$, $a = 0$, $\alpha = 0$, $\mathbf{y} = \mathbf{0}$, $\mathbf{z} = \mathbf{0}$, $\beta = 0$. These conditions can be checked in $O(1)$ time.

Introduce node: Suppose t is an introduction node with child t' such that $X_t = X_{t'} \cup \{v_i\}$ for some $v_i \notin X_{t'}$. Let A be any subset of X_t . We consider two cases:

Case (i): Let $v_i \notin A$. In this case $dp_t(A, \mathbf{x}, a, \alpha, \mathbf{y}, \mathbf{z}, \beta)$ is true if and only if $dp_{t'}(A, \mathbf{x}, a, \alpha, \mathbf{y}, \mathbf{z}, \beta)$ is true.

Case (ii): Let $v_i \in A$. Here $dp_t(A, \mathbf{x}, a, \alpha, \mathbf{y}, \mathbf{z}, \beta)$ is true if and only if there exist $A', \mathbf{x}', a', \alpha', \mathbf{y}', \mathbf{z}'$, and β' such that $dp_{t'}(A', \mathbf{x}', a', \alpha', \mathbf{y}', \mathbf{z}', \beta') = \text{true}$, where

1. $A = A' \cup \{v_i\}$;
2. $x(j) = x'(j) + 1$, if $v_j \in N_A(v_i)$, $x(i) = d_A(v_i)$, and $x(j) = x'(j)$ if $v_j \in A \setminus N_A[v_i]$;

3. $a = a' + 1$;
4. $\alpha = \alpha' + \delta$; here δ is the cardinality of the set

$$\left\{ v_j \in A \mid x'(j) < \frac{d_G(v_j) - 1}{2}; x(j) \geq \frac{d_G(v_j) - 1}{2} \right\}.$$

That is, to compute α from α' we need to add the number δ of those vertices not satisfied in $(A', \mathbf{x}', a', \alpha', \mathbf{y}', \mathbf{z}', \beta')$ but satisfied in $(A, \mathbf{x}, a, \alpha, \mathbf{y}, \mathbf{z}, \beta)$.

5. $y(i) = b$ and $y(j) = y'(j)$ for all $j \neq i$.
6. $z(S) = z'(S)$ if $v_i \notin S$; $z(S) = 0$ if $v_i \in S$.
7. $\beta = \beta'$.

For an introduce node t , $dp_t(A, \mathbf{x}, a, \alpha, \mathbf{y}, \mathbf{z}, \beta)$ can be computed in $O(1)$ time. This follows from the fact that there is only one candidate of such tuple $(A', \mathbf{x}', a', \alpha', \mathbf{y}', \mathbf{z}', \beta')$.

Forget node: Suppose t is a forget node with child t' such that $X_t = X_{t'} \setminus \{v_i\}$ for some $v_i \in X_{t'}$. Let A be any subset of X_t . Here $dp_t(A, \mathbf{x}, a, \alpha, \mathbf{y}, \mathbf{z}, \beta)$ is true if and only if either $dp_{t'}(A, \mathbf{x}, a, \alpha, \mathbf{y}, \mathbf{z}, \beta)$ is true (this corresponds to the case that A_t does not contain v_i) or $dp_{t'}(A', \mathbf{x}', a', \alpha', \mathbf{y}', \mathbf{z}', \beta') = \text{true}$ for some $A', \mathbf{x}', a', \alpha', \mathbf{y}', \mathbf{z}', \beta'$ with the following conditions (this corresponds to the case that A_t contains v_i):

1. $A' = A \cup \{v_i\}$;
2. $x(j) = x'(j)$ for all $j \neq i$ and $x(i) = 0$;
3. $a = a'$;
4. $\alpha = \alpha'$;

We now consider four cases:

Case 1: v_i is not marginally protected and v_i is a good vertex.

5. $y(j) = y'(j)$ for all $j \neq i$ and $y(i) = 0$;

6. $z(S) = z'(S)$ for all $S \subseteq A$;

7. $\beta = \beta'$.

Case 2: v_i is not marginally protected and v_i is a bad vertex.

5. $y(j) = y'(j)$ for all $j \neq i$ and $y(i) = 0$;

6.

$$z(S) = \begin{cases} z'(S) + 1 & \text{if } S \subseteq N_A(v_i) \\ z'(S) & \text{otherwise} \end{cases}$$

7. $\beta = \beta'$.

Case 3: v_i is marginally protected and v_i is a good vertex.

5.

$$y(j) = \begin{cases} g & \text{if } v_j \in N_A(v_i) \\ y'(j) & \text{if } v_j \in A \setminus N_A(v_i) \end{cases}$$

6. $z(S) = z'(S) - z'(S \cup \{v_i\})$ for all $S \subseteq A$;

7. $\beta = \beta' + z'(\{v_i\}) + |\{j : y'(j) = b; y(j) = g\}|$.

Case 4: v_i is marginally protected and v_i is a bad vertex.

5.

$$y(j) = \begin{cases} g & \text{if } v_j \in N_A(v_i) \\ y'(j) & \text{if } v_j \in A \setminus N_A(v_i) \end{cases}$$

6.

$$z(S) = \begin{cases} z'(S) - z'(S \cup \{v_i\}) + 1 & \text{if } S \subseteq N_A(v_i) \\ z'(S) - z'(S \cup \{v_i\}) & \text{for all other subsets } S \subseteq A \end{cases}$$

7. $\beta = \beta' + z'(\{v_i\}) + |\{j : y'(j) = b; y(j) = g\}|$.

For a forget node t , $dp_t(A, \mathbf{x}, a, \alpha, \mathbf{y}, \mathbf{z}, \beta)$ can be computed in $n^{O(2^k)}$ time. This follows from the fact that there are $n^{O(2^k)}$ candidates of such tuple $(A', \mathbf{x}', a', \alpha', \mathbf{z}', \beta')$, and each of them can be checked in $O(1)$ time.

Join node: Suppose t is a join node with children t_1 and t_2 such that $X_t = X_{t_1} = X_{t_2}$. Let A be any subset of X_t . Then $dp_t(A, \mathbf{x}, a, \alpha, \mathbf{y}, \mathbf{z}, \beta)$ is true if and only if there exist $(A_1, \mathbf{x}_1, a_1, \alpha_1, \mathbf{y}_1, \mathbf{z}_1, \beta_1)$ and $(A_2, \mathbf{x}_2, a_2, \alpha_2, \mathbf{y}_2, \mathbf{z}_2, \beta_2)$ such that $dp_{t_1}(A_1, \mathbf{x}_1, a_1, \alpha_1, \mathbf{y}_1, \mathbf{z}_1, \beta_1) = \text{true}$ and $dp_{t_2}(A_2, \mathbf{x}_2, a_2, \alpha_2, \mathbf{y}_2, \mathbf{z}_2, \beta_2) = \text{true}$, where

1. $A = A_1 = A_2$;
2. $x(i) = x_1(i) + x_2(i) - d_A(v_i)$ for all $i \in A$, and $x(i) = 0$ if $i \notin A$;
3. $a = a_1 + a_2 - |A|$;
4. $\alpha = \alpha_1 + \alpha_2 - \gamma + \delta$; γ is the cardinality of the set

$$\left\{ v_j \in A \mid x_1(j) \geq \frac{d_G(v_i) - 1}{2}; x_2(j) \geq \frac{d_G(v_i) - 1}{2} \right\}$$

and δ is the cardinality of the set

$$\left\{ v_j \in A \mid x_1(j) < \frac{d_G(v_i) - 1}{2}; x_2(j) < \frac{d_G(v_i) - 1}{2}; x(j) \geq \frac{d_G(v_i) - 1}{2} \right\}.$$

To compute α from $\alpha_1 + \alpha_2$, we need to subtract the number of those v_j which are satisfied in both the branches and add the number of vertices v_j not satisfied in either of the branches t_1 and t_1 but satisfied in t .

5.

$$y(j) = \begin{cases} g & \text{if } y_1(j) = g \text{ or } y_2(j) = g \\ b & \text{otherwise} \end{cases}$$

6. $z(S) = z_1(S) + z_2(S)$ for all $S \subseteq A$;
7. $\beta = \beta_1 + \beta_2 - |\{j : y_1(j) = g, y_2(j) = g\}|$.

For a join node t , there are n^k possible pairs for $(\mathbf{x}_1, \mathbf{x}_2)$ as \mathbf{x}_2 is uniquely determined by \mathbf{x}_1 ; $n + 1$ possible pairs for (a_1, a_2) ; $n + 1$ possible pairs for (α_1, α_2) ; there are 2^k possible

pairs for $(\mathbf{y}_1, \mathbf{y}_2)$ as \mathbf{y}_2 is uniquely determined by \mathbf{y}_1 ; there are n^{2^k} possible pairs for $(\mathbf{z}_1, \mathbf{z}_2)$ as \mathbf{z}_2 is uniquely determined by \mathbf{z}_1 ; and $n + 1$ possible pairs for (β_1, β_2) . In total, there are $2^k n^{O(2^k)}$ candidates, and each of them can be checked in $O(1)$ time. Thus, for a join node t , $dp_t(A, \mathbf{x}, a, \alpha, \mathbf{y}, \mathbf{z}, \beta)$ can be computed in $2^k n^{O(2^k)}$ time.

At the root node r , we look at all records such that $dp_r(\emptyset, \mathbf{x}, a, \alpha, \mathbf{y}, \mathbf{z}, \beta) = \text{true}$, and $a = \alpha = \beta$. The size of a maximum locally minimal defensive alliance is the maximum a satisfying $dp_r(\emptyset, \mathbf{x}, a, a, \mathbf{y}, \mathbf{z}, a) = \text{true}$.

Chapter 6

The Satisfactory Partition Problem

6.1 Introduction

In this chapter, we consider the SATISFACTORY PARTITION and BALANCED SATISFACTORY PARTITION problems. In the SATISFACTORY PARTITION problem, given a graph $G = (V, E)$, the goal is to find a nontrivial partition (V_1, V_2) of V such that for every $v \in V$, if $v \in V_i$ then $d_{V_i}(v) \geq d_{V_{3-i}}(v)$. In the BALANCED SATISFACTORY PARTITION problem, given a graph $G = (V, E)$, the goal is to find a nontrivial partition (V_1, V_2) of V such that $|V_1| = |V_2|$ and for every $v \in V$, if $v \in V_i$ then $d_{V_i}(v) \geq d_{V_{3-i}}(v)$. This chapter is based on the papers [28, 23, 24]. We design a polynomial-time algorithm for the SATISFACTORY PARTITION problem for block graphs. We prove that the SATISFACTORY PARTITION and BALANCED SATISFACTORY PARTITION problems are fixed parameter tractable (FPT) when parameterized by neighbourhood diversity. We show that the SATISFACTORY PARTITION and BALANCED SATISFACTORY PARTITION problems can be solved in polynomial time for graphs of bounded clique-width. Finally we prove that the BALANCED SATISFACTORY PARTITION problem is W[1]-hard when parameterized by treewidth.

6.2 Polynomial Time Algorithm for Block Graphs

A *cut-vertex* is a vertex the removal of which would disconnect the remaining graph. A graph G is a *block graph* if every block (maximal 2-connected subgraph) is a clique. By their maximality, different blocks of G overlap in at most one vertex, which is then a cut-vertex of G . Hence, every edge of G lies in a unique block, and G is the union of its blocks. An end block of a block graph is a block that contains exactly one cut-vertex of G . A block graph that is not complete graph has at least two end blocks. A block graph G is shown in Figure 6.1.

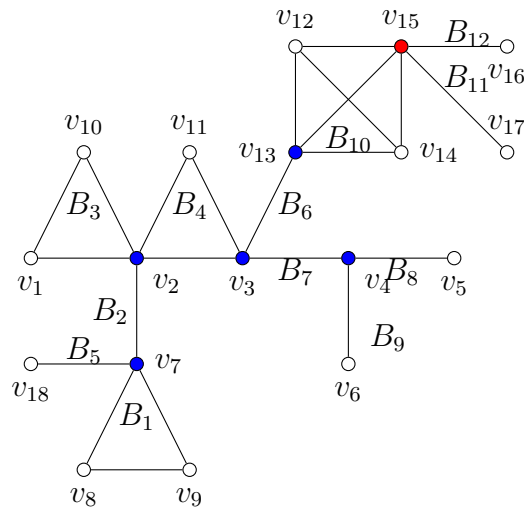


Figure 6.1: A block graph G . It has six cut-vertices; the good cut-vertices $v_2, v_3, v_4, v_7, v_{13}$ are shown in blue; and the bad cut-vertex v_{15} is shown in red.

Lemma 13. If a cut-vertex v belongs to an end block $B' = \{v, v'\}$ of size two, then v and v' must lie in the same part in any satisfactory partition.

Proof. Suppose v and v' are in different parts. As B' is an end-block of size 2, v' has no neighbour in its own part and has one neighbour v in the other part. Hence v' is not satisfied. This proves the lemma.

Let G be a block graph. Every cut-vertex of G belongs to at least two blocks. For simplicity,

suppose cut-vertex v belongs to two blocks B_1 and B_2 . Then $G - v$ has two components C_{B_1} and C_{B_2} , where C_{B_i} is the component that contains block $B_i \setminus \{v\}$, $i = 1, 2$. There are two possible partitions with respect to v : $(C_{B_1} \cup \{v\}, C_{B_2})$ and $(C_{B_1}, C_{B_2} \cup \{v\})$. A cut-vertex v is said to be a *good cut-vertex* if there is a satisfactory partition of G with respect to v ; otherwise v is called a *bad cut-vertex*. In general, suppose cut-vertex v belongs to k non-end blocks B_1, B_2, \dots, B_k , and ℓ end blocks $B'_1, B'_2, \dots, B'_\ell$. As each cut-vertex must belong to at least two blocks, we have $k + \ell \geq 2$. We consider the following cases and decide if v is a *good* or *bad* cut-vertex in each case.

Case 1: Let $k \geq 2$ and $\ell \geq 0$. That is, v belongs to at least two non-end blocks and $\ell \geq 0$ end blocks. Let B be the smallest non-end block containing v . Consider the partition $V_1 = C_B$ and $V_2 = V \setminus V_1$. Note that $v \in V_2$ and $d_{V_2}(v) \geq d_{V_1}(v)$, hence v is satisfied and, clearly all other vertices are also satisfied. Thus (V_1, V_2) forms a satisfactory partition. For example in Figure 6.1, v_2 belongs to two non-end blocks B_2, B_4 and one end block B_3 . Here B_2 is the smallest non-end block, thus $V_1 = C_{B_2} = \{v_7, v_8, v_9, v_{18}\}$ and $V_2 = V \setminus V_1$ form a satisfactory partition. The cut-vertex v_2 is a good cut-vertex.

Case 2: Let $k = 1$ and $\ell \geq 1$. That is, v belongs to exactly one non-end block B and at least one end block.

Subcase 2A: Suppose all the end blocks $B'_1, B'_2, \dots, B'_\ell$ are of size two. Using Lemma 13, we know the vertices of blocks $B'_1, B'_2, \dots, B'_\ell$ are in one part along with v . Thus $\bigcup_{i=1}^{\ell} B'_i$ forms the first part V_1 and $V_2 = V \setminus V_1$. Thus vertex $v \in V_1$ and $d_{V_1}(v) = \ell$. If the size of block B is greater than or equal to $\ell + 2$, v is not satisfied in partition (V_1, V_2) , as $d_{V_2}(v) \geq \ell + 1 > d_{V_1}(v) = \ell$. For example in Figure 6.1, v_{15} is adjacent to one non-end block B_{10} of size 4 and two end blocks B_{11}, B_{12} of size 2 each. Consider the partition with respect to v_{15} , $V_1 = \{v_{15}, v_{16}, v_{17}\}$ and $V_2 = V \setminus V_1$. This is not a satisfactory partition as $v_{15} \in V_1$ but $d_{V_2}(v_{15}) = 3 > d_{V_1}(v_{15}) = 2$. The cut-vertex v_{15} here is a bad cut-vertex. If the size of block B is less than or equal to $\ell + 1$, v is satisfied in partition (V_1, V_2) , as $d_{V_1}(v) = \ell \geq d_{V_2}(v)$. For example, v_4 is adjacent to one non-end block B_7 of size 2 and two end blocks B_8, B_9 of size 2 each. Note that partition with respect to v_4 , $V_1 = \{v_4, v_5, v_6\}$ and $V_2 = V \setminus V_1$, is a satisfactory partition. The cut-vertex v_4 here is a good cut-vertex.

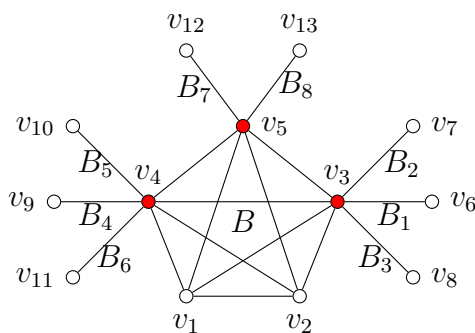


Figure 6.2: A satisfactory partitionable block graph $G = (V, E)$ with no good cut-vertices. Note that G has three cut-vertices v_3, v_4, v_5 and all of them are bad cut-vertices. Here $V_1 = \{v_3, v_4, v_6, v_7, v_8, v_9, v_{10}, v_{11}\}$ and $V_2 = V \setminus V_1$ form a satisfactory partition of G .

Subcase 2B: At least one end block is of size greater than 2. Without loss of generality suppose $|B'_1| > 2$. If $|B| \geq |B'_1|$, then $V_1 = C_{B'_1}$ and $V_2 = V \setminus V_1$ form a satisfactory partition. If $|B'_1| \geq |B|$, then $V_1 = C_B$ and $V_2 = V \setminus V_1$ form a satisfactory partition. For example, v_7 belongs to one non-end block $B_2 = \{v_2, v_7\}$ and two end blocks B_1 and B_5 . As the size of the non-end block B_2 is less than or equal to the size of end block B_1 , we set $V_1 = C_{B_2}$, the component that contains $B_2 \setminus \{v_7\}$ in $G - v_7$, and $V_2 = V \setminus V_1 = \{v_7, v_8, v_9, v_{18}\}$. The partition (V_1, V_2) forms a satisfactory partition. The cut-vertex v_7 is a good cut-vertex.

This suggests the following theorem.

Theorem 6.2.1. Let G be a block graph. If G has a good cut-vertex then G is satisfactory partitionable.

Note that although the condition of this theorem is sufficient to assure that a block graph is satisfactory partitionable, this certainly is not a necessary condition. For example, the block graph shown in Figure 6.2 is satisfactory partitionable but does not have any good cut-vertices; clearly such block graphs always have at least two bad cut-vertices. If a block graph has exactly one cut-vertex and that too is a bad cut-vertex, then the graph has no satisfactory partition. Now, we consider block graphs G having no good cut-vertices but the number m of bad cut-vertices is at least two; such graphs satisfy following two conditions:

1. There is exactly one non-end block in G and every cut-vertex belongs to it.
2. All the end blocks of G are of size exactly equal to 2.

Suppose B is the only non-end block in G and B' is obtained from B by removing its cut vertices. Let D_i represent the union of all end-blocks that contain v_i . For graph G in Figure 6.2, we have $D_3 = \{v_3, v_7, v_6, v_8\}$, $D_4 = \{v_4, v_9, v_{10}, v_{11}\}$, $D_5 = \{v_5, v_{12}, v_{13}\}$; and $B = \{v_1, v_2, v_3, v_4, v_5\}$, $B' = \{v_1, v_2\}$. By Lemma 13, all the vertices of D_i must lie in one part in any satisfactory partition. Let (D_1, D_2, \dots, D_m) be a decreasing ordering of D_i 's according to cardinalities, that is, $|D_1| \geq |D_2| \geq \dots \geq |D_m|$.

Theorem 6.2.2. Let G be a block graph satisfying Conditions 1 and 2 above. Then G is satisfactory partitionable if and only if G has a satisfactory partition of the form either

$$V_{1,r} = \bigcup_{i=1}^r D_i, \quad V_{2,r} = B' \cup \bigcup_{i=r+1}^m D_i \quad \text{or}$$

$$V'_{1,r} = B' \cup \bigcup_{i=1}^r D_i, \quad V'_{2,r} = \bigcup_{i=r+1}^m D_i,$$

for some $1 \leq r \leq m$.

Proof. Suppose G is satisfactory partitionable, and $V_1 = \bigcup_{i=1}^{j-1} D_i \cup \bigcup_{i=j+1}^{r+1} D_i$, $V_2 = B' \cup D_j \cup \bigcup_{i=r+2}^m D_i$ form a satisfactory partition of G . It is easy to see that $V'_1 = \bigcup_{i=1}^r D_i$ and $V'_2 = B' \cup \bigcup_{i=r+1}^m D_i$, obtained from (V_1, V_2) by swapping two sets D_j and D_{r+1} , also form a satisfactory partition in the required form. On the other hand, if there is a satisfactory partition of the form $(V_{1,r}, V_{2,r})$ or $(V'_{1,r}, V'_{2,r})$ for some r , then G is satisfactory partitionable. This proves the theorem.

The following algorithm determines if a given block graph G has a satisfactory partition.
SP-BLOCK GRAPH (G)

1. for each cut-vertex $v \in V(G)$, decide if v is a good cut-vertex or a bad cut-vertex.

2. if G has a good cut-vertex, then G is satisfactory partitionable (Theorem 6.2.1).
3. if G has exactly one bad cut-vertex and no good cut-vertices, then G is not satisfactory partitionable.
4. if G has at least two bad cut-vertices and no good cut-vertices, then compute all partitions of the form $(V_{1,r}, V_{2,r})$ and $(V'_{1,r}, V'_{2,r})$ for $1 \leq r \leq m$, and G is satisfactory partitionable if $(V_{1,r}, V_{2,r})$ or $(V'_{1,r}, V'_{2,r})$ is a satisfactory partition for some $1 \leq r \leq m$ (Theorem 6.2.2).

The total cost of deciding if v is a good or bad cut-vertex for all cut-vertices $v \in V(G)$, is $O(|E|)$. Computation of all partitions of the form $(V_{1,r}, V_{2,r})$ and $(V'_{1,r}, V'_{2,r})$ for $1 \leq r \leq m$, requires a decreasing ordering (D_1, D_2, \dots, D_m) of D_i 's according to their cardinalities. This takes $O(|V| \log |V|)$ time as m can be at most $|V|$. The running time of SP-BLOCK GRAPH is therefore $O(|V| \log |V| + |E|)$.

6.3 FPT algorithm parameterized by neighbourhood diversity

In this section, we present an FPT algorithm for the SATISFACTORY PARTITION and BALANCED SATISFACTORY PARTITION problems parameterized by neighbourhood diversity. See Section 1.7 for the definitions of neighbourhood diversity and type graph. In this section, we prove the following theorem:

Theorem 6.3.1. *The SATISFACTORY PARTITION problem is fixed-parameter tractable when parameterized by the neighbourhood diversity.*

Let G be a connected graph such that $\text{nd}(G) = k$. Let C_1, \dots, C_k be the partition of $V(G)$ into sets of type classes. We assume $k \geq 2$ since otherwise the problem becomes trivial. Let H be the type graph of G . We define $I_1 = \{C_i \mid C_i \subseteq V_1\}$, $I_2 = \{C_i \mid C_i \subseteq V_2\}$ and $I_3 = \{C_i \mid C_i \cap V_1 \neq \emptyset, C_i \cap V_2 \neq \emptyset\}$ where (V_1, V_2) is a satisfactory partition. We next guess if C_i belongs to I_1 , I_2 , or I_3 . There are at most 3^k possibilities as each C_i has three options: either in I_1 , I_2 , or I_3 . We reduce the problem of finding a satisfactory partition to an integer linear programming optimization with k variables. Since integer linear programming is fixed

parameter tractable when parameterized by the number of variables [47], we conclude that our problem is FPT when parameterized by the neighbourhood diversity.

ILP Formulation: Given I_1, I_2 and I_3 , our goal here is to answer if there exists a satisfactory partition (V_1, V_2) of G with all vertices of C_i are in V_1 if $C_i \in I_1$, all vertices of C_i are in V_2 if $C_i \in I_2$, and vertices of C_i are distributed amongst V_1 and V_2 if $C_i \in I_3$. For each C_i , we associate a variable: x_i that indicates $|V_1 \cap C_i| = x_i$. Because the vertices in C_i have the same neighbourhood, the variables x_i determine (V_1, V_2) uniquely, up to isomorphism. We now characterize a satisfactory partition in terms of x_i . Note that $x_i = n_i = |C_i|$ if $C_i \in I_1$; $x_i = 0$ if $C_i \in I_2$.

Lemma 14. *Let C be a clique type class. Then C is either in I_1 or I_2 .*

Proof. Let C be a clique type class. Let $u, v \in C$. Let us denote $N(u) \setminus \{v\} = N(v) \setminus \{u\}$ by S and let $a = |S \cap V_1|$ and let $b = |S \cap V_2|$. The satisfiability of u implies $a \geq b + 1$ and the satisfiability of v implies $b \geq a + 1$. Clearly, u and v cannot be satisfied simultaneously, as the two inequalities imply $a \geq b + 1 \geq a + 2$, a contradiction. This proves the lemma.

For $w = 1, 2, 3$ and $j = 1, 2, \dots, k$, we denote by $\mathcal{C}[j, w]$ the set of indices i such that $C_i \in N_H[C_j] \cap I_w$. Here $N_H[C_j]$ denotes the closed neighbourhood of node C_j in the type graph H . Now we consider the following four cases:

Case 1: Suppose v belongs to a clique type class C_j in I_1 . Then the degree of v in V_1 satisfies

$$d_{V_1}(v) = \sum_{i \in \mathcal{C}[j, 1]} n_i + \sum_{i \in \mathcal{C}[j, 3]} x_i - 1.$$

The degree of v in V_2 satisfies

$$d_{V_2}(v) = \sum_{i \in \mathcal{C}[j, 2]} n_i + \sum_{i \in \mathcal{C}[j, 3]} (n_i - x_i).$$

Therefore, vertex v is satisfied if and only if

$$\sum_{i \in \mathcal{C}[j, 1]} n_i + \sum_{i \in \mathcal{C}[j, 3]} 2x_i \geq 1 + \sum_{i \in \mathcal{C}[j, 2] \cup \mathcal{C}[j, 3]} n_i \quad (6.1)$$

Case 2: Suppose v belongs to a clique type class C_j in I_2 . Then similarly, v is satisfied if and only if

$$\sum_{i \in \mathcal{C}[j,2] \cup \mathcal{C}[j,3]} n_i \geq 1 + \sum_{i \in \mathcal{C}[j,1]} n_i + \sum_{i \in \mathcal{C}[j,3]} 2x_i \quad (6.2)$$

Case 3: For $w = 1, 2, 3$ and $j = 1, 2, \dots, k$, we denote by $\mathcal{C}(j, w)$ the set of indices i such that $C_i \in N_H(C_j) \cap I_w$. Here $N_H(C_j)$ denotes the open neighbourhood of node C_j in the type graph H . Suppose v belongs to an independent type class C_j in $I_1 \cup I_3$. Then the degree of v in V_1 satisfies

$$d_{V_1}(v) = \sum_{i \in \mathcal{C}(j,1)} n_i + \sum_{i \in \mathcal{C}(j,3)} x_i.$$

Note that if $C_j \in I_3$, then only x_j vertices of C_j are in V_1 and the remaining y_j vertices of C_j are in V_2 . The degree of v in V_2 satisfies

$$d_{V_2}(v) = \sum_{i \in \mathcal{C}(j,2)} n_i + \sum_{i \in \mathcal{C}(j,3)} (n_i - x_i).$$

Therefore, v is satisfied if and only if

$$\sum_{i \in \mathcal{C}(j,1)} n_i + \sum_{i \in \mathcal{C}(j,3)} 2x_i \geq \sum_{i \in \mathcal{C}(j,2) \cup \mathcal{C}(j,3)} n_i \quad (6.3)$$

Case 4: Suppose v belongs to an independent type class C_j in $I_2 \cup I_3$. Similarly, vertex v is satisfied if and only if

$$\sum_{i \in \mathcal{C}(j,2) \cup \mathcal{C}(j,3)} n_i \geq \sum_{i \in \mathcal{C}(j,1)} n_i + \sum_{i \in \mathcal{C}(j,3)} 2x_i \quad (6.4)$$

We now formulate ILP formulation of satisfactory partition, for given I_1, I_2 and I_3 . The question is whether there exist x_j under the conditions $x_j = n_j$ if $C_j \in I_1$, $x_j = 0$ if $C_j \in I_2$, $x_j \in \{1, 2, \dots, n_j - 1\}$ if $C_j \in I_3$ and the additional conditions described below:

- Inequality 6.1 for all clique type classes $C_j \in I_1$
- Inequality 6.2 for all clique type classes $C_j \in I_2$

- Inequality 6.3 for all independent type classes $C_j \in I_1$
- Inequality 6.4 for all independent type classes $C_j \in I_2$
-

$$\sum_{i \in \mathcal{C}(j,2) \cup \mathcal{C}(j,3)} n_i = \sum_{i \in \mathcal{C}(j,1)} n_i + \sum_{i \in \mathcal{C}(j,3)} 2x_i$$

for all independent type classes $C_j \in I_3$.

For BALANCED SATISFACTORY PARTITION problem, we additionally ask that

$$\sum_{i: C_i \in I_1} n_i + \sum_{i: C_i \in I_3} x_i = \sum_{i: C_i \in I_3} (n_i - x_i) + \sum_{i: C_i \in I_2} n_i.$$

Solving the ILP: In the formulation for SATISFACTORY PARTITION problem, we have at most k variables. The value of any variable in the integer linear programming is bounded by n , the number of vertices in the input graph. The constraints can be represented using $O(k^2 \log n)$ bits. Lemma 1 implies that we can solve the problem with the given guess I_1, I_2 and I_3 in FPT time. There are at most 3^k choices for (I_1, I_2, I_3) , and the ILP formula for a guess can be solved in FPT time. Thus Theorem 6.3.1 holds.

6.4 Graphs of bounded clique-width

This section presents a polynomial time algorithm for the SATISFACTORY PARTITION and BALANCED SATISFACTORY PARTITION problems for graphs of bounded clique-width. See Section 1.7 for the definition of clique-width and irredundant c -expression. We now have the following result:

Theorem 6.4.1. *Given an n -vertex graph G and an irredundant c -expression T of G , the SATISFACTORY PARTITION and BALANCED SATISFACTORY PARTITION problems are solvable in $O(n^{8c})$ time.*

For each node t in a c -expression T , let G_t be the vertex-labeled graph represented by subtree hanging from t . We denote by V_t the vertex set of G_t . In a vertex-labeled graph, an

i -vertex is a vertex of label i . For each i , we denote the set of i -vertices in G_t by V_t^i . For each node t in T , we construct a table $dp_t(\mathbf{r}_1, \mathbf{r}_2, \mathbf{s}_1, \mathbf{s}_2) \in \{\text{true}, \text{false}\}$ where $\mathbf{r}_1, \mathbf{r}_2, \mathbf{s}_1$ and \mathbf{s}_2 are c -dimensional vectors. We set $dp_t(\mathbf{r}_1, \mathbf{r}_2, \mathbf{s}_1, \mathbf{s}_2) = \text{true}$ if and only if there exists a partition (V_1, V_2) of V_t such that for all $i \in \{1, 2, \dots, c\}$

- $\mathbf{r}_1(i) = |V_1 \cap V_t^i|$;
- $\mathbf{r}_2(i) = |V_2 \cap V_t^i|$;
- if $V_1 \cap V_t^i \neq \emptyset$, then $\mathbf{s}_1(i) = \min_{v \in V_1 \cap V_t^i} \{|N_{G_t}(v) \cap V_1| - |N_{G_t}(v) \cap V_2|\}$, otherwise $\mathbf{s}_1(i) = \infty$;
- if $V_2 \cap V_t^i \neq \emptyset$, then $\mathbf{s}_2(i) = \min_{v \in V_2 \cap V_t^i} \{|N_{G_t}(v) \cap V_2| - |N_{G_t}(v) \cap V_1|\}$, otherwise $\mathbf{s}_2(i) = \infty$.

That is, $\mathbf{r}_1(i)$ denotes the number of the i -vertices in V_1 ; $\mathbf{r}_2(i)$ denotes the number of the i -vertices in V_2 ; $\mathbf{s}_1(i)$ is the ‘‘surplus’’ at the weakest i -vertex in V_1 and $\mathbf{s}_2(i)$ is the ‘‘surplus’’ at the weakest i -vertex in V_2 . The possible values that $\mathbf{r}_1(i)$ and $\mathbf{r}_2(i)$ can take are from the set $\{0, \dots, n\}$; the possible values that $\mathbf{s}_1(i)$ and $\mathbf{s}_2(i)$ can take are from the set $\{-n + 1, \dots, n - 1\} \cup \{\infty\}$.

Let τ be the root of the c -expression T of G . Then G has a satisfactory partition if there exist $\mathbf{r}_1, \mathbf{r}_2, \mathbf{s}_1, \mathbf{s}_2$ satisfying

1. $dp_\tau(\mathbf{r}_1, \mathbf{r}_2, \mathbf{s}_1, \mathbf{s}_2) = \text{true}$;
2. $\min\{\mathbf{s}_1(i), \mathbf{s}_2(i)\} \geq 0$.

For the BALANCED SATISFACTORY PARTITION problem, we additionally ask that $\sum_{i=1}^c \mathbf{r}_1(i) = \sum_{i=1}^c \mathbf{r}_2(i)$. If all entries $dp_t(\mathbf{r}_1, \mathbf{r}_2, \mathbf{s}_1, \mathbf{s}_2)$ are computed in advance, then we can verify above conditions by spending $O(1)$ time for each tuple $(\mathbf{r}_1, \mathbf{r}_2, \mathbf{s}_1, \mathbf{s}_2)$.

In the following, we compute all entries $dp_t(\mathbf{r}_1, \mathbf{r}_2, \mathbf{s}_1, \mathbf{s}_2)$ in a bottom-up manner. There are $(n + 1)^c \cdot (n + 1)^c \cdot (2n)^c \cdot (2n)^c = O(n^{4c})$ possible tuples $(\mathbf{r}_1, \mathbf{r}_2, \mathbf{s}_1, \mathbf{s}_2)$. Thus, to prove Theorem 6.4.1, it is enough to prove that each entry $dp_t(\mathbf{r}_1, \mathbf{r}_2, \mathbf{s}_1, \mathbf{s}_2)$ can be computed in time $O(n^{4c})$ assuming that the entries for the children of t are already computed.

Lemma 15. For a leaf node t with label i , $dp_t(\mathbf{r}_1, \mathbf{r}_2, \mathbf{s}_1, \mathbf{s}_2)$ can be computed in $O(1)$ time.

Proof. Observe that $dp_t(\mathbf{r}_1, \mathbf{r}_2, \mathbf{s}_1, \mathbf{s}_2) = \mathbf{true}$ if and only if $\mathbf{r}_1(j) = 0$, $\mathbf{r}_2(j) = 0$, $\mathbf{s}_1(j) = 0$, and $\mathbf{s}_2(j) = 0$ for all $j \neq i$ and either

- $\mathbf{r}_1(i) = 0$, $\mathbf{r}_2(i) = 1$, $\mathbf{s}_1(i) = \infty$, $\mathbf{s}_2(i) = 0$, or
- $\mathbf{r}_1(i) = 1$, $\mathbf{r}_2(i) = 0$, $\mathbf{s}_1(i) = 0$, $\mathbf{s}_2(i) = \infty$.

The first case corresponds to $V_1 = \emptyset, V_2 = V_t^i$, and the second case corresponds to $V_1 = V_t^i, V_2 = \emptyset$. These conditions can be checked in $O(1)$ time.

Lemma 16. For a \cup -node t , $dp_t(\mathbf{r}_1, \mathbf{r}_2, \mathbf{s}_1, \mathbf{s}_2)$ can be computed in $O(n^{4c})$ time.

Proof. Let t' and t'' be the children of t in T . Then $dp_t(\mathbf{r}_1, \mathbf{r}_2, \mathbf{s}_1, \mathbf{s}_2) = \mathbf{true}$ if and only if there exist $\mathbf{r}'_1, \mathbf{r}'_2, \mathbf{s}'_1, \mathbf{s}'_2$ and $\mathbf{r}''_1, \mathbf{r}''_2, \mathbf{s}''_1, \mathbf{s}''_2$ such that $dp_{t'}(\mathbf{r}'_1, \mathbf{r}'_2, \mathbf{s}'_1, \mathbf{s}'_2) = \mathbf{true}$, $dp_{t''}(\mathbf{r}''_1, \mathbf{r}''_2, \mathbf{s}''_1, \mathbf{s}''_2) = \mathbf{true}$, $\mathbf{r}_1(i) = \mathbf{r}'_1(i) + \mathbf{r}''_1(i)$, $\mathbf{r}_2(i) = \mathbf{r}'_2(i) + \mathbf{r}''_2(i)$, $\mathbf{s}_1(i) = \min\{\mathbf{s}'_1(i), \mathbf{s}''_1(i)\}$ and $\mathbf{s}_2(i) = \min\{\mathbf{s}'_2(i), \mathbf{s}''_2(i)\}$ for all i . The number of possible pairs for $(\mathbf{r}'_1, \mathbf{r}''_1)$ is at most $(n+1)^c$ as \mathbf{r}''_1 is uniquely determined by \mathbf{r}'_1 ; the number of possible pairs for $(\mathbf{r}'_2, \mathbf{r}''_2)$ is at most $(n+1)^c$ as \mathbf{r}''_2 is uniquely determined by \mathbf{r}'_2 . There are at most $2^c(2n)^c$ possible pairs for $(\mathbf{s}'_1, \mathbf{s}''_1)$ and for $(\mathbf{s}'_2, \mathbf{s}''_2)$ each. In total, there are $O(n^{4c})$ candidates. Each candidate can be checked in $O(1)$ time, thus the lemma holds.

Lemma 17. For a η_{ij} -node t , $dp_t(\mathbf{r}_1, \mathbf{r}_2, \mathbf{s}_1, \mathbf{s}_2)$ can be computed in $O(1)$ time.

Proof. Let t' be the child of t in T . Then, $dp_t(\mathbf{r}_1, \mathbf{r}_2, \mathbf{s}_1, \mathbf{s}_2) = \mathbf{true}$ if and only if $dp_{t'}(\mathbf{r}_1, \mathbf{r}_2, \mathbf{s}'_1, \mathbf{s}'_2) = \mathbf{true}$ for some $\mathbf{s}'_1, \mathbf{s}'_2$ with the following conditions:

- $\mathbf{s}_1(h) = \mathbf{s}'_1(h)$ and $\mathbf{s}_2(h) = \mathbf{s}'_2(h)$ hold for all $h \notin \{i, j\}$;
- $\mathbf{s}_1(i) = \mathbf{s}'_1(i) + 2\mathbf{r}_1(j) - |V_t^j|$ and $\mathbf{s}_1(j) = \mathbf{s}'_1(j) + 2\mathbf{r}_1(i) - |V_t^i|$;
- $\mathbf{s}_2(i) = \mathbf{s}'_2(i) + 2\mathbf{r}_2(j) - |V_t^j|$ and $\mathbf{s}_2(j) = \mathbf{s}'_2(j) + 2\mathbf{r}_2(i) - |V_t^i|$.

We now explain the condition for $\mathbf{s}_1(i)$. Recall that T is irredundant. That is, the graph $G_{t'}$ does not have any edge between the i -vertices and the j -vertices. In G_t , an i -vertex has

exactly $\mathbf{r}_1(j)$ more neighbours in V_1 and exactly $|V_t^j| - \mathbf{r}_1(j)$ more neighbours in V_2 . Thus we have $\mathbf{s}_1(i) = \mathbf{s}'_1(i) + 2\mathbf{r}_1(j) - |V_t^j|$. The lemma holds as there is only one candidate for each $\mathbf{s}'_1(i)$, $\mathbf{s}'_1(j)$, $\mathbf{s}'_2(i)$ and $\mathbf{s}'_2(j)$.

Lemma 18. For a $\rho_{i \rightarrow j}$ node t , $dp_t(\mathbf{r}_1, \mathbf{r}_2, \mathbf{s}_1, \mathbf{s}_2)$ can be computed in $O(n^4)$ time.

Proof. Let t' be the child of t in T . Then, $dp_t(\mathbf{r}_1, \mathbf{r}_2, \mathbf{s}_1, \mathbf{s}_2) = \text{true}$ if and only if there exist $\mathbf{r}'_1, \mathbf{r}'_2, \mathbf{s}'_1, \mathbf{s}'_2$ such that $dp_{t'}(\mathbf{r}'_1, \mathbf{r}'_2, \mathbf{s}'_1, \mathbf{s}'_2) = \text{true}$, where :

- $\mathbf{r}_1(i) = 0$, $\mathbf{r}_1(j) = \mathbf{r}'_1(i) + \mathbf{r}'_1(j)$, and $\mathbf{r}_1(h) = \mathbf{r}'_1(h)$ if $h \notin \{i, j\}$;
- $\mathbf{r}_2(i) = 0$, $\mathbf{r}_2(j) = \mathbf{r}'_2(i) + \mathbf{r}'_2(j)$, and $\mathbf{r}_2(h) = \mathbf{r}'_2(h)$ if $h \notin \{i, j\}$;
- $\mathbf{s}_1(i) = \infty$, $\mathbf{s}_1(j) = \min\{\mathbf{s}'_1(i), \mathbf{s}'_1(j)\}$, and $\mathbf{s}_1(h) = \mathbf{s}'_1(h)$ if $h \notin \{i, j\}$;
- $\mathbf{s}_2(i) = \infty$, $\mathbf{s}_2(j) = \min\{\mathbf{s}'_2(i), \mathbf{s}'_2(j)\}$, and $\mathbf{s}_2(h) = \mathbf{s}'_2(h)$ if $h \notin \{i, j\}$.

The number of possible pairs for $(\mathbf{r}'_1(i), \mathbf{r}'_1(j))$ is $O(n)$ as $\mathbf{r}'_1(j)$ is uniquely determined by $\mathbf{r}'_1(i)$; similarly the number of possible pairs for $(\mathbf{r}'_2(i), \mathbf{r}'_2(j))$ is $O(n)$ as $\mathbf{r}'_2(j)$ is uniquely determined by $\mathbf{r}'_2(i)$. There are at most $O(n)$ possible pairs for $(\mathbf{s}'_1(i), \mathbf{s}'_1(j))$ and for $(\mathbf{s}'_2(i), \mathbf{s}'_2(j))$. In total, there are $O(n^4)$ candidates. Each candidate can be checked in $O(1)$ time, thus the lemma holds.

6.5 Hardness of BALANCED SATISFACTORY PARTITION parameterized by treewidth

In this section, we prove the following theorem:

Theorem 6.5.1. The BALANCED SATISFACTORY PARTITION problem is W[1]-hard when parameterized by the treewidth of the graph.

We introduce several variants of BALANCED SATISFACTORY PARTITION that we require in our proofs. The problem BALANCED SATISFACTORY PARTITION^S generalizes BALANCED SATISFACTORY PARTITION where some vertices are forced to be in the second part V_2 . This

variant can be formalized as follows:

BALANCED SATISFACTORY PARTITION^S
Input: A graph $G = (V, E)$ on an even number of vertices, and a set $V_{\square} \subseteq V(G)$.
Question: Is there a balanced satisfactory partition (V_1, V_2) of V such that $V_{\square} \subseteq V_2$.

BALANCED SATISFACTORY PARTITION^{FS} is a further generalization where some vertices are forced to be in the first part V_1 and some other vertices are forced to be in the second part V_2 . This variant can be formalized as follows:

BALANCED SATISFACTORY PARTITION^{FS}
Input: A graph $G = (V, E)$ on an even number of vertices, a set $V_{\Delta} \subseteq V(G)$, and a set $V_{\square} \subseteq V(G)$.
Question: Is there a balanced satisfactory partition (V_1, V_2) of V such that (i) $V_{\Delta} \subseteq V_1$ (ii) $V_{\square} \subseteq V_2$.

Finally, we introduce the generalization BALANCED SATISFACTORY PARTITION^{FSC} in which we are also given a subset of "complementary pairs" of vertices and feasible solutions are only those for which neither V_1 nor V_2 contains both the vertices of a complementary pair.

BALANCED SATISFACTORY PARTITION^{FSC}
Input: A graph $G = (V, E)$ on an even number of vertices, a set $V_{\Delta} \subseteq V(G)$, a set $V_{\square} \subseteq V(G)$, and a set $C \subseteq V(G) \times V(G)$.
Question: Is there a balanced satisfactory partition (V_1, V_2) of V such that (i) $V_{\Delta} \subseteq V_1$ (ii) $V_{\square} \subseteq V_2$, and (iii) for all $(a, b) \in C$, V_1 contains either a or b but not both?

Let $G = (V, E)$ be an undirected and edge weighted graph, where V , E , and w denote the set of nodes, the set of edges and a positive integral weight function $w : E \rightarrow Z^+$, respectively. An orientation Λ of G is an assignment of a direction to each edge $(u, v) \in E(G)$, that is, either (u, v) or (v, u) is contained in Λ . The weighted outdegree of u on Λ is $w_{\text{out}}^u = \sum_{(u, v) \in \Lambda} w(u, v)$. We define MINIMUM MAXIMUM OUTDEGREE problem as follows:

MINIMUM MAXIMUM OUTDEGREE

Input: A graph G , an edge weighting w of G given in unary, and a positive integer r .

Question: Is there an orientation Λ of G such that $w_{\text{out}}^u \leq r$ for each $u \in V(G)$?

It is known that MINIMUM MAXIMUM OUTDEGREE is $W[1]$ -hard when parameterized by the treewidth of the input graph [57]. To prove Theorem 6.5.1, we reduce MINIMUM MAXIMUM OUTDEGREE to BALANCED SATISFACTORY PARTITION^{FSC} and then show how we can successively reduce the latter problem to BALANCED SATISFACTORY PARTITION^{FS}, BALANCED SATISFACTORY PARTITION^S and finally BALANCED SATISFACTORY PARTITION. To measure the treewidth of a BALANCED SATISFACTORY PARTITION^{FSC} instance, we use the following definition. Let $I = (G, V_\Delta, V_\square, C)$ be a BALANCED SATISFACTORY PARTITION^{FSC} instance. The *primal graph* G' of I is defined as follows: $V(G') = V(G)$ and $E(G') = E(G) \cup C$.

6.5.1 Hardness of BALANCED SATISFACTORY PARTITION with Restriction on the First Part (F), the Second Part (S) and Complementary Pairs

To show $W[1]$ -hardness of BALANCED SATISFACTORY PARTITION^{FSC}, we reduce from MINIMUM MAXIMUM OUTDEGREE, which is known to be $W[1]$ -hard parameterized by the treewidth of the graph [57]:

Lemma 19. *The BALANCED SATISFACTORY PARTITION^{FSC} is $W[1]$ -hard when parameterized by the treewidth of the primal graph.*

Proof. Let $G = (V, E, w)$ and a positive integer r be an instance of MINIMUM MAXIMUM OUTDEGREE. We construct an instance of BALANCED SATISFACTORY PARTITION^{FSC} as follows. An example is given in Figure 6.3. For each vertex $v \in V(G)$, we introduce a set of new vertices $H_v = \{h_1^v, \dots, h_{2r}^v\}$. For each edge $(u, v) \in E(G)$, we introduce the sets of new vertices $V_{uv} = \{u_1^v, \dots, u_{w(u,v)}^v\}$, $V'_{uv} = \{u_1^{v'}, \dots, u_{w(u,v)}^{v'}\}$, $V_{vu} = \{v_1^u, \dots, v_{w(u,v)}^u\}$, $V'_{vu} = \{v_1^{u'}, \dots, v_{w(u,v)}^{u'}\}$, $V_{uv}^\square = \{u_1^{\square}, \dots, u_{w(u,v)}^{\square}\}$, $V'_{uv}^\square = \{u_1^{v'\square}, \dots, u_{w(u,v)}^{v'\square}\}$, $V_{vu}^\square = \{v_1^{u\square}, \dots, v_{w(u,v)}^{u\square}\}$, and $V'_{vu}^\square = \{v_1^{u'\square}, \dots, v_{w(u,v)}^{u'\square}\}$. Let $\omega = \sum_{(u,v) \in E(G)} w(u, v)$. Finally we add a set V_0 of $8\omega +$

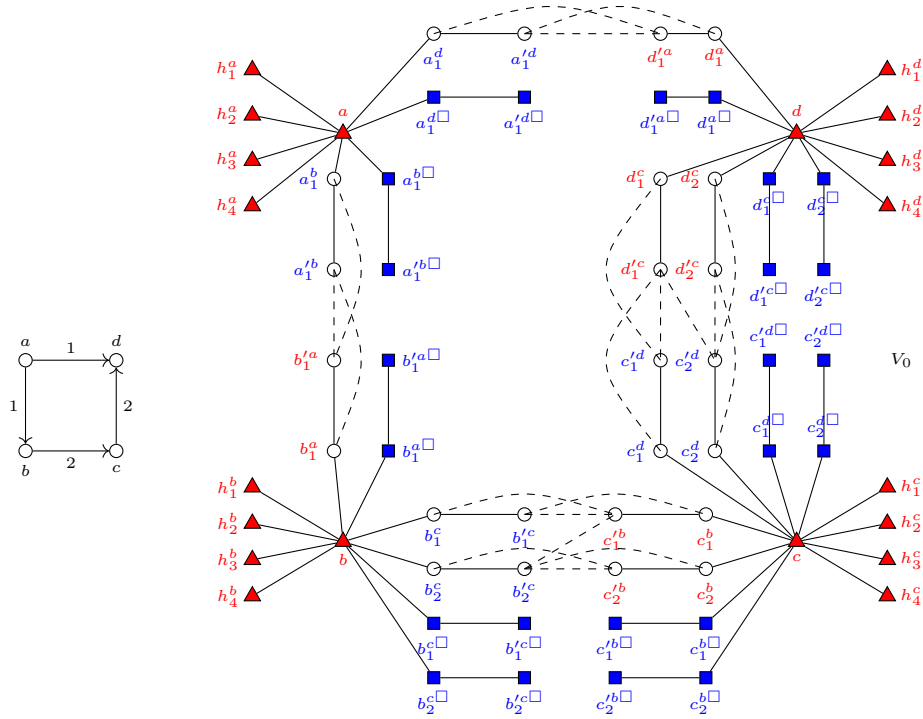


Figure 6.3: Result of our reduction on a MINIMUM MAXIMUM OUTDEGREE instance G with $r = 2$. The graph G long with its orientation is shown at the left; and G' is shown at the right. Complementary vertex pairs are shown using dashed lines. The vertices of the set V_Δ are filled with red color whereas the vertices of the set V_\square are filled with blue color. The vertices in the first part of satisfactory partition (V_1, V_2) of G' are shown in red label and vertices of V_2 are shown in blue label for the given orientation of G . Here $\omega = 6$ and V_0 contains 64 isolated vertices.

$|V|(2r + 1) - 4$ isolated vertices. We now define the graph G' with

$$\begin{aligned}
 V(G') = & V(G) \cup V_0 \cup \bigcup_{v \in V(G)} H_v \cup \bigcup_{(u,v) \in E(G)} (V_{uv} \cup V_{uv}^\square \cup V_{vu} \cup V_{vu}^\square) \\
 & \cup \bigcup_{(u,v) \in E(G)} (V'_{uv} \cup V'_{uv}^\square \cup V'_{vu} \cup V'_{vu}^\square)
 \end{aligned}$$

and

$$\begin{aligned}
E(G') = & \{(v, h) \mid v \in V(G), h \in H_v\} \cup \{(u, x) \mid (u, v) \in E(G), x \in V_{uv} \cup V_{uv}^\square\} \\
& \cup \{(x, v) \mid (u, v) \in E(G), x \in V_{vu} \cup V_{vu}^\square\} \\
& \cup \{(u_i^v, u_i'^v), (u_i^{v\square}, u_i'^{v\square}), (v_i^u, v_i'^u), (v_i^{u\square}, v_i'^{u\square}) \mid (u, v) \in E(G), 1 \leq i \leq w(u, v)\}.
\end{aligned}$$

The number of vertices in $V(G') \setminus V_0$ is $8\omega + |V|(2r + 1)$. We define the complementary vertex pairs

$$\begin{aligned}
C = & \{(u_i'^v, v_i'^u), (u_i^v, v_i'^u), (u_i'^v, v_i^u) \mid (u, v) \in E(G), 1 \leq i \leq w(u, v)\} \\
& \cup \{(u_{i+1}^v, v_i'^u) \mid (u, v) \in E(G), 1 \leq i < w(u, v)\}
\end{aligned}$$

Complementary vertex pairs are shown in dashed lines in Figure 6.3. Finally we define

$$V_\Delta = V(G) \cup \bigcup_{v \in V(G)} H_v$$

and

$$V_\square = \bigcup_{(u,v) \in E(G)} (V_{uv}^\square \cup V_{uv}'^\square \cup V_{vu}^\square \cup V_{vu}'^\square).$$

We use I to denote $(G', V_\Delta, V_\square, C)$ which is an instance of BALANCED SATISFACTORY PARTITION^{FSC}.

Clearly, it takes polynomial time to compute I . We now prove that the treewidth of the primal graph G' of I is bounded by a function of the treewidth of G . We do so by modifying an optimal tree decomposition τ of G as follows:

- For every edge (u, v) of G , there is a node in τ whose bag B contains both u and v ; add to this node a chain of nodes $1, 2, \dots, w(u, v) - 1$ where the bag of node i is $B \cup \{u_i^v, u_i'^v, v_i'^u, v_i^u, u_{i+1}^v, u_{i+1}'^v, v_{i+1}'^u, v_{i+1}^u\}$.
- For every edge (u, v) of G , there is a node in τ whose bag B contains u ; add to this node a chain of nodes $1, 2, \dots, w(u, v)$ where the bag of node i is $B \cup \{u_i^{v\square}, u_i'^{v\square}\}$.
- For every edge (u, v) of G , there is a node in τ whose bag B contains v and add to this

node a chain of nodes $1, 2, \dots, w(u, v)$ where the bag of node i is $B \cup \{v_i^{u\Box}, v_i'^{u\Box}\}$.

- For every vertex v of G , there is a node in τ whose bag B contains v and add to this node a chain of nodes $1, 2, \dots, 2r$ where the bag of node i is $B \cup \{h_i^v\}$.

Clearly, the modified tree decomposition is a valid tree decomposition of the primal graph of I and its width is at most the treewidth of G plus eight.

It remains to show that our reduction is correct. Let D be the directed graph obtained by an orientation of the edges of G such that for each vertex the sum of the weights of outgoing edges is at most r . Note that $(u, v) \in E(D)$ indicates there is a directed edge from u to v in D whereas $(u, v) \in E(G)$ indicates there is an edge joining u and v in G . Consider the partition of $G' - V_0$

$$V_1 = V_\Delta \cup \bigcup_{(u,v) \in E(D)} (V_{vu} \cup V'_{vu}) = V(G) \cup \bigcup_{v \in V(G)} H_v \cup \bigcup_{(u,v) \in E(D)} (V_{vu} \cup V'_{vu})$$

and

$$\begin{aligned} V_2 &= V_\square \cup \bigcup_{(u,v) \in E(D)} (V_{uv} \cup V'_{uv}) \\ &= \bigcup_{(u,v) \in E(D)} (V_{uv} \cup V'_{uv} \cup V_{uv}^\square \cup V'_{uv}^\square) \cup \bigcup_{(u,v) \in E(D)} (V_{vu}^\square \cup V'_{vu}^\square). \end{aligned}$$

To prove that (V_1, V_2) is a satisfactory partition, first we prove that $d_{V_1}(x) \geq d_{V_2}(x)$ for all $x \in V_1$. If x is a vertex in H_v or $V_{vu} \cup V'_{vu}$, then clearly all neighbours of x are in V_1 , hence x is satisfied. Suppose $x \in V(G)$. Let w_{out}^x and w_{in}^x denote the sum of the weights of outgoing and incoming edges of vertex x , respectively. Hence $d_{V_1}(x) = 2r + w_{\text{in}}^x$ and $d_{V_2}(x) = 2w_{\text{out}}^x + w_{\text{in}}^x$ in G' . This shows that x is satisfied as $w_{\text{out}}^x \leq r$. Now we prove that $d_{V_2}(x) \geq d_{V_1}(x)$ for all $x \in V_2$. If x is a vertex in $V_{uv} \cup V_{uv}^\square \cup V'_{vu} \cup V'_{vu}^\square$ then x has one neighbour in V_1 and one neighbour in V_2 . If $x \in V'_{uv} \cup V'_{uv}^\square \cup V'_{vu} \cup V'_{vu}^\square$ then x has one neighbour in V_2 and no neighbours in V_1 . Thus the vertices in V_2 are satisfied. The isolated vertices of V_0 are distributed among V_1 and V_2 so that it becomes balanced satisfactory partition for G' .

Conversely, suppose (V_1, V_2) is a balanced satisfactory partition of G' . That is $|V_1| = |V_2| = 8\omega + (2r + 1)|V| - 2$. Then $V'_1 = V_1 \setminus V_0$ and $V'_2 = V_2 \setminus V_0$ form a satisfactory partition of $G' - V_0$. For every $(u, v) \in E(G)$, either $V_{uv} \cup V'_{uv} \in V'_1$ or $V_{vu} \cup V'_{vu} \in V'_1$ due to the

complementary vertex pairs. We define a directed graph D by $V(D) = V(G)$ and

$$E(D) = \left\{ (u, v) \mid V_{vu} \cup V'_{vu} \in V'_1 \right\} \cup \left\{ (v, u) \mid V_{uv} \cup V'_{uv} \in V'_1 \right\}.$$

Suppose there is a vertex x in D for which $w_{\text{out}}^x > r$. Clearly $x \in V'_1$. We know $d_{V'_1}(x) = 2r + w_{\text{in}}^x$ and $d_{V'_2}(x) = 2w_{\text{out}}^x + w_{\text{in}}^x$. Then $d_{V'_2}(x) > d_{V'_1}(x)$, as by assumption $w_{\text{out}}^x > r$, a contradiction to the fact that (V'_1, V'_2) is a satisfactory partition of $G' - V_0$. Hence $w_{\text{out}}^x \leq r$ for all $x \in V(D)$. \square

6.5.2 Hardness of BALANCED SATISFACTORY PARTITION with Restriction on the First Part (F) and the Second Part (S)

Now we prove the following result which eliminates complementary pairs.

Lemma 20. *The BALANCED SATISFACTORY PARTITION^{FS} problem, parameterized by the treewidth of the graph, is $W[1]$ -hard.*

Proof. Let $I = (G, V_{\square}, V_{\Delta}, C)$ be an instance of BALANCED SATISFACTORY PARTITION^{FSC}. Consider the primal graph of I , that is the graph G^p where $V(G^p) = V(G)$ and $E(G^p) = E(G) \cup C$. From this we construct an instance $I' = (G', V'_{\square}, V'_{\Delta})$ of BALANCED SATISFACTORY PARTITION^{FS} problem. For each $(a, b) \in C$ in the primal graph G^p , we introduce two new vertices Δ^{ab} and \square^{ab} and four new edges in G' . We now define G' with

$$V(G') = V(G) \cup \bigcup_{(a,b) \in C} \{\Delta^{ab}, \square^{ab}\}$$

and

$$E(G') = E(G) \cup \bigcup_{(a,b) \in C} \{(a, \Delta^{ab}), (a, \square^{ab}), (b, \Delta^{ab}), (b, \square^{ab})\}.$$

Finally, we define the sets

$$V'_{\Delta} = V_{\Delta} \cup \bigcup_{(a,b) \in C} \{\Delta^{ab}\}$$

and

$$V'_\square = V_\square \cup \bigcup_{(a,b) \in C} \{\square^{ab}\}.$$

We illustrate our construction in Figure 6.4. It is easy to see that we can compute I' in polynomial time and its treewidth is linear in the treewidth of I .

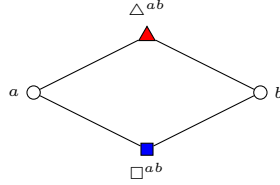


Figure 6.4: Gadget for a pair of complementary vertices (a, b) in the reduction from BALANCED SATISFACTORY PARTITION^{FSC} to BALANCED SATISFACTORY PARTITION^{FS}.

The following holds for every solution (V'_1, V'_2) of I' : V'_1 contains Δ^{ab} for every $(a, b) \in C$, so it must also contain a or b . It cannot contain both a and b for any $(a, b) \in C$, because $\square^{ab} \in V'_2$. Restricting (V'_1, V'_2) to the original vertices thus is a solution to I . Conversely, for every solution (V_1, V_2) of I , the partition (V'_1, V'_2) where $V'_1 = V_1 \cup \bigcup_{(a,b) \in C} \{\Delta^{ab}\}$ and $V'_2 = V_2 \cup \bigcup_{(a,b) \in C} \{\square^{ab}\}$, is a solution of I' . \square

6.5.3 Hardness of BALANCED SATISFACTORY PARTITION with Restriction on the Second Part (S)

Now we prove the following result which eliminates restriction on the first part of the partition.

Lemma 21. *The BALANCED SATISFACTORY PARTITION^S is $W[1]$ -hard when parameterized by the treewidth of the graph.*

Proof. Let $I = (G, V_\Delta, V_\square)$ be a BALANCED SATISFACTORY PARTITION^{FS} instance; let n

denote the number of vertices in G . Fix any vertex $v_0 \in V_\square$. For every $u \in V_\Delta$, we introduce two sets of new vertices $X_u = \{x_1^u, x_2^u, \dots, x_n^u\}$ and $Y_u^\square = \{y_1^u, y_2^u, \dots, y_n^u\}$. Next, we define the BALANCED SATISFACTORY PARTITION^S instance $I' = (G', V'_\square)$ where $V'_\square = V_\square \cup \bigcup_{u \in V_\Delta} Y_u^\square$ and G' is the graph defined by

$$V(G') = V(G) \cup \bigcup_{u \in V_\Delta} X_u \cup Y_u^\square$$

and

$$\begin{aligned} E(G') = & E(G) \cup \bigcup_{u \in V_\Delta} \left\{ (u, x_i^u), (u, y_i^u), (x_i^u, v_0), (y_i^u, v_0) \mid 1 \leq i \leq n \right\} \\ & \cup \bigcup_{u \in V_\Delta} \left\{ (x_i^u, y_i^u), (x_i^u, y_{i+1}^u) \mid 1 \leq i \leq n-1 \right\} \cup \left\{ (x_n^u, y_n^u), (x_n^u, y_1^u) \right\} \\ & \cup \bigcup_{u \in V_\Delta} \left\{ (x_i^u, x_{i+1}^u) \mid 1 \leq i \leq n-1 \right\} \cup \left\{ (x_n^u, x_1^u) \right\} \\ & \cup \bigcup_{u \in V_\Delta} \left\{ (y_i^u, y_{i+1}^u) \mid 1 \leq i \leq n-1 \right\} \cup \left\{ (y_n^u, y_1^u) \right\} \end{aligned}$$

An example is given in Figure 6.5. The treewidth of G' is equal to the treewidth of G plus

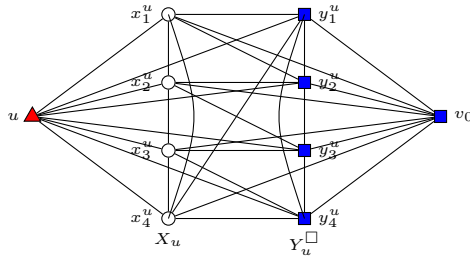


Figure 6.5: Let $n = 4$. Gadget for a pair of vertices (u, v_0) where $u \in V_\Delta$ and v_0 is a fixed vertex in V_\square in the reduction from BALANCED SATISFACTORY PARTITION^{FS} to BALANCED SATISFACTORY PARTITION^S.

5. We now claim that I is a yes-instance if and only if I' is a yes-instance. Assume first that there exists a balanced satisfactory partition (V_1, V_2) of I such that $V_\Delta \in V_1$ and $V_\square \in V_2$.

In this case, we get a balanced satisfactory partition (V'_1, V'_2) of I' as follows:

$$V'_1 = V_1 \cup \bigcup_{u \in V_\Delta} X_u \quad \text{and} \quad V'_2 = V_2 \cup \bigcup_{u \in V_\Delta} Y_u^\square.$$

It is easy to see that (V'_1, V'_2) forms a balanced satisfactory partition of G' as all the vertices in V_1 and V_2 remain satisfied and also the new vertices in $X_u \cup Y_u^\square$ for all $u \in V_\Delta$ are satisfied in their respective part as each vertex has three neighbours in its own part and three neighbors in the other part. Since we are adding equal number of vertices in the balanced partition (V_1, V_2) , we again get a balanced satisfactory partition. This shows that I' is a yes-instance.

Conversely, suppose that there exists a balanced satisfactory partition (V'_1, V'_2) of G' such that $V'_1 \subseteq V'_2$. We first show that all the vertices in V_Δ must lie in V'_1 . Let us assume that there exists a vertex $u \in V_\Delta$ that lies in V'_2 . Then each vertex in X_u has at least 4 neighbors in V'_2 and at most 2 neighbours in V'_1 ; therefore all the vertices in X_u lie in V'_2 . In this case, we cannot get a balanced satisfactory partition as already more than half of the vertices are in V'_2 . This proves that all the vertices in V_Δ lie in V'_1 . Next, we show that as $V_\Delta \subseteq V'_1$, the vertices in $\bigcup_{u \in V_\Delta} X_u$ also lie in V'_1 . Since $u \in V'_1$, it must be satisfied in V'_1 . As the vertices in Y_u^\square lie in V'_2 , u has at least n neighbors in V'_2 and since u has at most $n-1$ neighbors in graph G , it implies that at least one vertex from X_u must be in V'_1 . Without loss of generality, we can assume that $x_1^u \in V'_1$. Since $x_1^u \in V'_1$, it must be satisfied in V'_1 and this forces its neighbours x_n^u, x_2^u to be in V'_1 as well. Repetitively applying the above argument we get that all the vertices in set X_u lie in V'_1 . We claim that $(V'_1 \cap V(G), V'_2 \cap V(G))$ forms a balanced satisfactory partition of graph G . As each vertex in $V'_i \cap V(G)$, $i = 1, 2$, loses equal number of neighbors from both the partitions, this implies that all the vertices are satisfied and the partition is balanced. This shows that I is a yes-instance. \square

6.5.4 Hardness of BALANCED SATISFACTORY PARTITION

Now we prove the following result which eliminates restriction on the second part of the partition.

Lemma 22. The BALANCED SATISFACTORY PARTITION problem, parameterized by the treewidth of the graph, is W[1]-hard.

Proof. Let $I = (G, V_\square)$ be a BALANCED SATISFACTORY PARTITION^S instance, where $V_\square = \{u_1, u_2, \dots, u_r\}$. For every vertex u_i in the set V_\square , we introduce two new sets of vertices $X^{u_i} = \{x_1^{u_i}, x_2^{u_i}, \dots, x_{4n}^{u_i}\}$ and $Y^{u_i} = \{y_1^{u_i}, y_2^{u_i}, \dots, y_{4n}^{u_i}\}$. We also introduce a clique of size 2 containing vertices $\{s, t\}$ and a set $C = \{c_1, c_2, \dots, c_{8n}\}$ of $8n$ vertices. We add two new vertices $\{s', t'\}$ along with two sets of vertices $S' = \{s'_1, s'_2, \dots, s'_{4n}\}$ and $T' = \{t'_1, t'_2, \dots, t'_{4n}\}$. Now, we define the BALANCED SATISFACTORY PARTITION instance $I' = G'$ where G' is the graph defined by

$$V(G') = V(G) \cup \{s, t, s', t'\} \cup S' \cup T' \cup C \cup \bigcup_{i=1}^r X^{u_i} \cup Y^{u_i}$$

and

$$\begin{aligned} E(G') = E(G) \cup & \bigcup_{i=1}^r \bigcup_{j=1}^{4n} \left\{ (x_j^{u_i}, u_i), (y_j^{u_i}, u_i), (y_j^{u_i}, s), (y_j^{u_i}, t) \right\} \cup \{(s, t)\} \\ & \cup \bigcup_{j=1}^{8n} \left\{ (c_j, s), (c_j, t) \right\} \cup \bigcup_{j=1}^{4n} \left\{ (s', s'_j), (t', t'_j) \right\} \cup \left\{ (s', s)(s', t), (t', s), (t', t) \right\}. \end{aligned}$$

We claim that I is a yes-instance of BALANCED SATISFACTORY PARTITION^S if and only if I' is a yes-instance of BALANCED SATISFACTORY PARTITION. Suppose that there is a balanced satisfactory partition (V_1, V_2) in G such that $V_\square \subseteq V_2$. A balanced satisfactory partition (V'_1, V'_2) for G' is defined as follows:

$$V'_1 = V_1 \cup \{s, t\} \cup C \cup \bigcup_{i=1}^r Y^{u_i} \text{ and } V'_2 = V_2 \cup \{s', t'\} \cup S' \cup T' \cup \bigcup_{i=1}^r X^{u_i}.$$

Clearly, all the vertices are satisfied. Since we are adding equal number of vertices in both the parts, (V'_1, V'_2) is a balanced satisfactory partition of G' . This proves that if I is a yes-

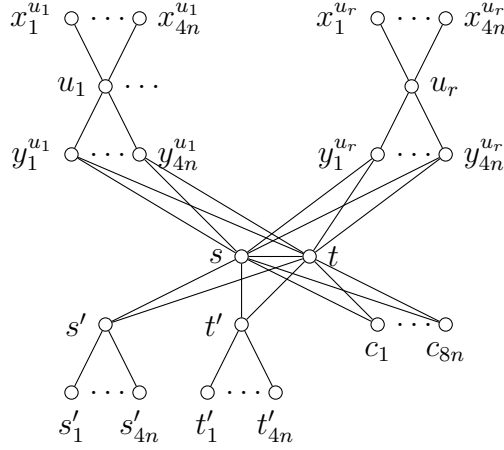


Figure 6.6: An illustration of the reduction from BALANCED SATISFACTORY PARTITION^S to BALANCED SATISFACTORY PARTITION.

instance then I' is a yes-instance.

To prove the reverse direction of the equivalence, suppose now that (V'_1, V'_2) is a balanced satisfactory partition of G' . We first prove that all the vertices of V_\square are in the same part. Since $N_{G'}[s] = N_{G'}[t]$, both s and t would be in the same part; without loss of generality suppose they lie in V'_1 . For $1 \leq i \leq r$, each vertex $y_j^{u_i}$ is adjacent to 3 vertices $\{u_i, s, t\}$ and since $\{s, t\}$ belong to V'_1 , it forces $y_j^{u_i}$ to be in V'_1 for $1 \leq j \leq 4n$. Similarly, as $s, t \in V'_1$, each c_i would also be in V'_1 for $1 \leq i \leq 8n$. For the sake of contradiction, suppose the vertices of V_\square are distributed among V'_1 and V'_2 , that is, r' many vertices of V_\square are in V'_1 and the remaining $r - r'$ vertices of V_\square are in V'_2 . Observe that if $u \in V_\square$ is in V'_1 , then all the vertices in X^u are also in V'_1 . This implies that V'_1 contains at least $4n(r + r' + 2) + r' + 2$ vertices and V'_2 contains at most $4n(r - r' + 2) + 2 + (r - r')$ vertices. It implies that $|V'_1| > |V'_2|$, a contradiction to our assumption that (V'_1, V'_2) is a balanced satisfactory partition of G' . This shows that all the vertices of V_\square must go to V'_2 . Therefore, for every balanced satisfactory partition of G' , we have

$$\{s, t\} \cup C \cup \bigcup_{i=1}^r Y^{u_i} \subseteq V'_1 \text{ and } \{s', t'\} \cup S' \cup T' \cup V_\square \cup \bigcup_{i=1}^r X^{u_i} \subseteq V'_2.$$

We now claim that $(V'_1 \cap V(G), V'_2 \cap V(G))$ forms a balanced satisfactory partition of G .

From the above observation, we have $V_{\square} \subset V'_2 \cap V(G)$. All the vertices are satisfied in the new partition $(V'_1 \cap V(G), V'_2 \cap V(G))$ and it is a balanced partition because we are removing equal number of vertices from both parts. This shows that if I' is a yes-instance then I is also a yes-instance.

This proves Theorem 6.5.1.

Chapter 7

Conclusions and Open Problems

In this dissertation, we have considered parameterized algorithms and complexity of the following graph problems: defensive and offensive alliances in graphs, locally minimal defensive alliances in graphs, and the satisfactory partition problem in graphs. In Chapter 3, we proved that `DEFENSIVE ALLIANCE` and `OFFENSIVE ALLIANCE` problems are FPT when parameterized by neighbourhood diversity; `DEFENSIVE ALLIANCE` and `OFFENSIVE ALLIANCE` problems are solvable in polynomial time on graphs of bounded treewidth. A powerful alliance is both defensive and offensive. The parameterized complexity of different kinds of alliances such as offensive alliance or powerful alliance remains unsettled when parameterized by clique-width, treewidth, pathwidth, feedback vertex set number, etc.

In Chapter 4, we have proved that the `DEFENSIVE ALLIANCE` problem is $W[1]$ -hard when parameterized by the pathwidth of the input graph and the `EXACT DEFENSIVE ALLIANCE` problem is $W[1]$ -hard parameterized by a wide range of fairly restrictive structural parameters such as the feedback vertex set number, pathwidth, treewidth and treedepth of the input graph. The parameterized complexity of the `DEFENSIVE ALLIANCE` problem remains unsettled when parameterized by the feedback vertex set number, pathwidth and treedepth of the input graph. It would also be interesting to consider the parameterized complexity with respect to twin-cover and modular width.

In Chapter 5, we have designed a polynomial-time algorithm for the `CONNECTED LOCALLY MINIMAL STRONG DEFENSIVE ALLIANCE` on trees. We proved that `LOCALLY MINIMAL DEFENSIVE ALLIANCE` problem is NP-complete, even when restricted to planar graphs.

We gave a randomized FPT algorithm for the EXACT CONNECTED LOCALLY MINIMAL DEFENSIVE ALLIANCE problem using color coding technique. We gave an FPT algorithm for LOCALLY MINIMAL DEFENSIVE ALLIANCE when parameterized by neighbourhood diversity of the input graph. We proved that EXACT CONNECTED LOCALLY MINIMAL DEFENSIVE ALLIANCE parameterized by treewidth is $W[1]$ -hard and thus not FPT (unless $FPT=W[1]$). Finally we showed that the LOCALLY MINIMAL DEFENSIVE ALLIANCE problem is polynomial time solvable for graphs with bounded treewidth. See Figure 7.1 for a schematic

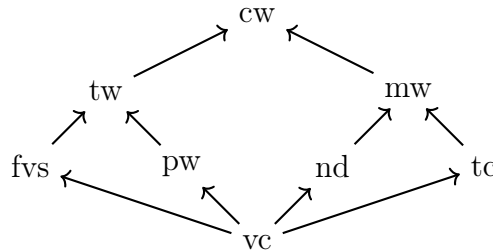


Figure 7.1: Relationship between vertex cover (vc), neighbourhood diversity (nd), twin cover (tc), modular width (mw), feedback vertex set (fvs), pathwidth (pw), treewidth (tw) and clique-width (cw). Arrow indicate generalizations, for example, treewidth generalizes both feedback vertex set and pathwidth.

representation of the relationship between selected graph parameters. Note that $A \rightarrow B$ means that there exists a function f such that for all graphs, $f(A(G)) \geq B(G)$; therefore the existence of an FPT algorithm parameterized by B implies the existence of an FPT algorithm parameterized by A , and conversely, any negative result parameterized by A implies the same negative result parameterized by B . We now list some nice problems emerge from the results here: is the LOCALLY MINIMAL DEFENSIVE ALLIANCE problem FPT in treewidth, and does it admit a polynomial kernel in neighborhood diversity? Also, noting that the result for neighborhood diversity implies that the problem is FPT in vertex cover, it would be interesting to consider the parameterized complexity with respect to twin cover. The modular width parameter also appears to be a natural parameter to consider here, and since there are graphs with bounded modular-width and unbounded neighborhood diversity; we believe this is also an interesting open problem. The parameterized complexity of the LOCALLY MINIMAL DEFENSIVE ALLIANCE problem remains unsettled when parameterized by other important structural graph parameters like clique-width, modular width etc.

In Chapter 6, we proved that the SATISFACTORY PARTITION problem is polynomial time solvable for block graphs; the SATISFACTORY PARTITION and BALANCED SATISFACTORY

PARTITION problems are fixed parameter tractable (FPT) when parameterized by neighbourhood diversity; the problems are polynomial time solvable for graphs of bounded clique width; and the BALANCED SATISFACTORY PARTITION problem is W[1]-hard when parameterized by treewidth. Noting that the result for neighborhood diversity implies that the problem is FPT in vertex cover, it would be interesting to consider the parameterized complexity with respect to twin cover. We have proved that the BALANCED SATISFACTORY PARTITION problem is W[1]-hard when parameterized by treewidth, but the parameterized complexity of SATISFACTORY PARTITION parameterized by treewidth remains unsettled. The parameterized complexity of SATISFACTORY PARTITION and BALANCED SATISFACTORY PARTITION remain unsettled when parameterized by other important structural graph parameters like pathwidth, clique-width and modular-width.

List of Publications:

The main results of this thesis are based on the following publications:

1. Ajinkya Gaikwad, Soumen Maity and Shuvam Kant Tripathi, Parameterized Complexity of Satisfactory Partition Problem, **Theoretical Computer Science**, Vol. 907, 113-127, 2022.
2. Ajinkya Gaikwad, Soumen Maity and Shuvam Kant Tripathi, Parameterized Complexity of Locally Minimal Defensive Alliances, **Discrete Applied Mathematics** (under revision).
3. Ajinkya Gaikwad, Soumen Maity and Shuvam Kant Tripathi, Parameterized intractability of defensive alliance problem, *In: 8th International Conference on Algorithms and Discrete Applied Mathematics, CALDAM 2022*, **Lecture Notes in Computer Science**, Vol. 13179, 279-291, 2022.
4. Ajinkya Gaikwad, Soumen Maity and Shuvam Kant Tripathi, The Balanced Satisfactory Partition Problem, *In: 47th International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2021*, **Lecture Notes in Computer Science**, Vol. 12607, 322-336, 2021.
5. Ajinkya Gaikwad, Soumen Maity and Shuvam Kant Tripathi, Parameterized Complexity of Locally Minimal Defensive Alliance, *In: 7th International Conference on Algorithms and Discrete Applied Mathematics, CALDAM 2021*, **Lecture Notes in Computer Science**, Vol. 12601, 135-148, 2021.
6. Ajinkya Gaikwad, Soumen Maity and Shuvam Kant Tripathi, Parameterized Complexity of Defensive and Offensive Alliances in Graphs, *In: 17th International Conference on Distributed Computing and Internet Technology, ICDCIT 2021*, **Lecture Notes in Computer Science**, Vol. 12582, 175-187, 2021.
7. Ajinkya Gaikwad, Soumen Maity and Shuvam Kant Tripathi, Parameterized Complexity of Satisfactory Partition Problem, *In: 14th International Conference on Combinatorial Optimization and Applications, COCOA 2020*, **Lecture Notes in Computer Science**, Vol. 12577, 76-90, 2020.

Bibliography

- [1] C. Bazgan, H. Fernau, and Z. Tuza. Aspects of upper defensive alliances. *Discrete Applied Mathematics*, 266:111 – 120, 2019.
- [2] C. Bazgan, Z. Tuza, and D. Vanderpooten. On the existence and determination of satisfactory partitions in a graph. In T. Ibaraki, N. Katoh, and H. Ono, editors, *Algorithms and Computation*, pages 444–453, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.
- [3] C. Bazgan, Z. Tuza, and D. Vanderpooten. Complexity and approximation of satisfactory partition problems. In *Proceedings of the 11th Annual International Conference on Computing and Combinatorics - Volume 3595*, page 829–838, Berlin, Heidelberg, 2005. Springer-Verlag.
- [4] C. Bazgan, Z. Tuza, and D. Vanderpooten. Degree-constrained decompositions of graphs: Bounded treewidth and planarity. *Theoretical Computer Science*, 355(3):389 – 395, 2006.
- [5] C. Bazgan, Z. Tuza, and D. Vanderpooten. The satisfactory partition problem. *Discrete Applied Mathematics*, 154(8):1236 – 1245, 2006.
- [6] B. Bliem and S. Woltran. Defensive alliances in graphs of bounded treewidth. *Discrete Applied Mathematics*, 251:334 – 339, 2018.
- [7] H. L. Bodlaender. A tourist guide through treewidth. *Acta Cybern.*, 11:1–21, 1993.
- [8] H. L. Bodlaender and J. Engelfriet. Domino treewidth. *Journal of Algorithms*, 24(1):94 – 123, 1997.
- [9] C.-W. Chang, M.-L. Chia, C.-J. Hsu, D. Kuo, L.-L. Lai, and F.-H. Wang. Global defensive alliances of trees and cartesian product of paths and cycles. *Discrete Applied Mathematics*, 160(4):479 – 487, 2012.
- [10] M. Chellali and T. W. Haynes. Global alliances and independence in trees. *Discuss. Math. Graph Theory*, 27(1):19–27, 2007.
- [11] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.

- [12] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 2012.
- [13] R. Enciso. *Alliances in graphs: Parameterized algorithms and on partitioning series-parallel graphs*. PhD thesis, USA, 2009.
- [14] M. R. Fellows, D. Lokshtanov, N. Misra, F. A. Rosamond, and S. Saurabh. Graph layout problems parameterized by vertex cover. In S.-H. Hong, H. Nagamochi, and T. Fukunaga, editors, *Algorithms and Computation*, pages 294–305, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.
- [15] M. R. Fellows, D. Lokshtanov, N. Misra, F. A. Rosamond, and S. Saurabh. Graph layout problems parameterized by vertex cover. In S.-H. Hong, H. Nagamochi, and T. Fukunaga, editors, *Algorithms and Computation*, pages 294–305, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.
- [16] H. Fernau and D. Raible. Alliances in graphs: a complexity-theoretic study. In J. van Leeuwen, G. F. Italiano, W. van der Hoek, C. Meinel, H. Sack, F. Plasil, and M. Bieliková, editors, *SOFSEM 2007: Theory and Practice of Computer Science, 33rd Conference on Current Trends in Theory and Practice of Computer Science, Harrachov, Czech Republic, January 20-26, 2007, Proceedings Volume II*, pages 61–70. Institute of Computer Science AS CR, Prague, 2007.
- [17] H. Fernau, J. A. Rodríguez-Velázquez, and J. M. Sigarreta. Global r-alliances and total domination. In *CTW*, 2008.
- [18] H. Fernau, J. A. Rodríguez, and J. M. Sigarreta. Offensive r-alliances in graphs. *Discrete Applied Mathematics*, 157(1):177 – 182, 2009.
- [19] G. Flake, S. Lawrence, and C. Giles. Efficient identification of web communities. In R. Ramakrishnan, S. Stolfo, R. Bayardo, I. Parsa, R. Ramakrishnan, S. Stolfo, R. Bayardo, and I. Parsa, editors, *Proceeding of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Proceeding of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 150–160, United States, 2000. Association for Computing Machinery (ACM). Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2001) ; Conference date: 20-08-2000 Through 23-08-2000.
- [20] G. W. Flake, S. Lawrence, and C. L. Giles. Efficient identification of web communities. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, page 150–160, New York, NY, USA, 2000. Association for Computing Machinery.
- [21] G. Fricke, L. Lawson, T. Haynes, M. Hedetniemi, and S. Hedetniemi. A note on defensive alliances in graphs. *Bulletin of the Institute of Combinatorics and its Applications*, 38:37–41, 2003.

- [22] A. Gaikwad and S. Maity. On structural parameterizations of the offensive alliance problem. *CoRR*, abs/2110.15757, 2021.
- [23] A. Gaikwad, S. Maity, and S. K. Tripathi. Parameterized complexity of satisfactory partition problem. In W. Wu and Z. Zhang, editors, *Combinatorial Optimization and Applications - 14th International Conference, COCOA 2020, Dallas, TX, USA, December 11-13, 2020, Proceedings*, volume 12577 of *Lecture Notes in Computer Science*, pages 76–90. Springer, 2020.
- [24] A. Gaikwad, S. Maity, and S. K. Tripathi. The balanced satisfactory partition problem. In T. Bures, R. Dondi, J. Gamper, G. Guerrini, T. Jurdzinski, C. Pahl, F. Sikora, and P. W. H. Wong, editors, *SOFSEM 2021: Theory and Practice of Computer Science - 47th International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2021, Bolzano-Bozen, Italy, January 25-29, 2021, Proceedings*, volume 12607 of *Lecture Notes in Computer Science*, pages 322–336. Springer, 2021.
- [25] A. Gaikwad, S. Maity, and S. K. Tripathi. Parameterized complexity of defensive and offensive alliances in graphs. In D. Goswami and T. A. Hoang, editors, *Distributed Computing and Internet Technology - 17th International Conference, ICDCIT 2021, Bhubaneswar, India, January 7-10, 2021, Proceedings*, volume 12582 of *Lecture Notes in Computer Science*, pages 175–187. Springer, 2021.
- [26] A. Gaikwad, S. Maity, and S. K. Tripathi. Parameterized complexity of locally minimal defensive alliances. In A. Mudgal and C. R. Subramanian, editors, *Algorithms and Discrete Applied Mathematics - 7th International Conference, CALDAM 2021, Rupnagar, India, February 11-13, 2021, Proceedings*, volume 12601 of *Lecture Notes in Computer Science*, pages 135–148. Springer, 2021.
- [27] A. Gaikwad, S. Maity, and S. K. Tripathi. Parameterized complexity of locally minimal defensive alliances. *CoRR*, abs/2105.10742, 2021.
- [28] A. Gaikwad, S. Maity, and S. K. Tripathi. Parameterized complexity of satisfactory partition problem. *Theoretical Computer Science*, 907:113–127, 2022.
- [29] A. Gaikwad, S. Maity, and S. K. Tripathi. Parameterized intractability of defensive alliance problem. In N. Balachandran and R. Inkulu, editors, *Algorithms and Discrete Applied Mathematics - 8th International Conference, CALDAM 2022, Puducherry, India, February 10-12, 2022, Proceedings*, volume 13179 of *Lecture Notes in Computer Science*, pages 279–291. Springer, 2022.
- [30] R. Ganian, F. Klute, and S. Ordyniak. On structural parameterizations of the bounded-degree vertex deletion problem. *Algorithmica*, 2020.
- [31] M. Gerber and D. Kobler. Classes of graphs that can be partitioned to satisfy all their vertices. *AUSTRALASIAN JOURNAL OF COMBINATORICS Volume*, 29:201–214, 01 2004.

- [32] M. U. Gerber and D. Kobler. Algorithmic approach to the satisfactory graph partitioning problem. *European Journal of Operational Research*, 125(2):283 – 291, 2000.
- [33] M. U. Gerber and D. Kobler. Algorithms for vertex-partitioning problems on graphs with fixed clique-width. *Theoretical Computer Science*, 299(1):719 – 734, 2003.
- [34] K. Hassan-Shafique. Partitioning a graph in alliances and its application to data clustering. 2004.
- [35] T. W. Haynes, S. T. Hedetniemi, and M. A. Henning. Global defensive alliances in graphs. *Electron. J. Comb.*, 10, 2003.
- [36] J. D. Horton and K. Kilakos. Minimum edge dominating sets. *SIAM Journal on Discrete Mathematics*, 6(3):375–387, 1993.
- [37] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.
- [38] L. H. Jamieson. *Algorithms and Complexity for Alliances and Weighted Alliances of Various Types*. PhD thesis, USA, 2007.
- [39] L. H. Jamieson, S. T. Hedetniemi, and A. A. McRae. The algorithmic complexity of alliances in graphs. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 68:137–150, 2009.
- [40] M. Kamiński, V. V. Lozin, and M. Milanič. Recent developments on graphs of bounded clique-width. *Discrete Applied Mathematics*, 157(12):2747 – 2761, 2009.
- [41] R. Kannan. Minkowski’s convex body theorem and integer programming. *Mathematics of Operations Research*, 12(3):415–440, 1987.
- [42] M. Kiyomi and Y. Otachi. Alliances in graphs of bounded clique-width. *Discrete Applied Mathematics*, 223:91 – 97, 2017.
- [43] T. Kloks. Treewidth, computations and approximations. In *Lecture Notes in Computer Science*, 1994.
- [44] T. Kloks. *Treewidth, Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer, 1994.
- [45] P. Kristiansen, M. Hedetniemi, and S. Hedetniemi. Alliances in graphs. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 48:157–177, 2004.
- [46] M. Lampis. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica*, 64:19–37, 2012.
- [47] H. W. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, 1983.

- [48] D. Manlove. In *Minimaximal and maximinimal optimisation problems: a partial order-based approach*, 1998.
- [49] J. Nešetřil and P. O. de Mendez. *Sparsity: Graphs, Structures, and Algorithms*. Springer Publishing Company, Incorporated, 2014.
- [50] N. Robertson and P. Seymour. Graph minors. iii. planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49 – 64, 1984.
- [51] J. Rodríguez-Velázquez and J. Sigarreta. Global offensive alliances in graphs. *Electronic Notes in Discrete Mathematics*, 25:157 – 164, 2006.
- [52] J. Sigarreta, S. Bermudo, and H. Fernau. On the complement graph and defensive k-alliances. *Discrete Applied Mathematics*, 157(8):1687 – 1695, 2009.
- [53] J. Sigarreta and J. Rodríguez. On defensive alliances and line graphs. *Applied Mathematics Letters*, 19(12):1345 – 1350, 2006.
- [54] J. Sigarreta and J. Rodríguez. On defensive alliances and line graphs. *Applied Mathematics Letters*, 19(12):1345–1350, 2006.
- [55] J. Sigarreta and J. Rodríguez. On the global offensive alliance number of a graph. *Discrete Applied Mathematics*, 157(2):219 – 226, 2009.
- [56] P. K. Srimani and Z. Xu. Distributed protocols for defensive and offensive alliances in network graphs using self-stabilization. In *2007 International Conference on Computing: Theory and Applications (ICCTA '07)*, pages 27–31, 2007.
- [57] S. Szeider. Not so easy problems for tree decomposable graphs. *CoRR*, abs/1107.1177, 2011.
- [58] M. Tedder, D. Corneil, M. Habib, and C. Paul. Simpler linear-time modular decomposition via recursive factorizing permutations. In *Automata, Languages and Programming*, pages 634–645, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.
- [59] M. Thorup. All structured programs have small tree width and good register allocation. *Information and Computation*, 142(2):159–181, 1998.
- [60] D. B. West. *Introduction to Graph Theory*. Prentice Hall, 2000.
- [61] M. Yannakakis and F. Gavril. Edge dominating sets in graphs. *SIAM Journal on Applied Mathematics*, 38(3):364–372, 1980.