Parameterized Complexity of the Hiding Leader Problem

A Thesis

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

by

Mohan Karra

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

October, 2022

Supervisor: Dr. Soumen Maity

Certificate

This is to certify that this dissertation entitled Parameterized Complexity of the Hiding Leader Problem towards the partial fulfilment of the BS-MS dual degree programme at the Indian Institute of Science Education and Research, Pune represents study/work carried out by Mohan Karra at Indian Institute of Science Education and Research under the supervision of Dr. Soumen Maity, Associate Professor, Department of Mathematics , during the academic year 2021-2022.

Dr. Soumen Maity

Committee:

Dr. Soumen Maity

Prof. Venkatesh Raman

This thesis is dedicated to Mental health

Declaration

I hereby declare that the matter embodied in the report entitled Parameterized Complexity of the Hiding Leader Problem are the results of the work carried out by me at the Department of Mathematics, Indian Institute of Science Education and Research, Pune, under the supervision of Dr. Soumen Maity and the same has not been submitted elsewhere for any other degree.

Mohan Karra

Acknowledgments

I want to start by expressing my gratitude to Dr. Soumen Maity for always being considerate and understanding. Without his kind assistance, this thesis would not be conceivable. This gratitude goes to Professor Venkatesh Raman, who has always been gracious. I was able to complete pivotal sections of this thesis with their advice.

IISER Pune has been a monumental step in my life. I've learned from it, developed a variety of talents, and met many new people. I am grateful to the IISER family for giving me the chance and resources to complete this thesis. Dr. Anindya Goswami and Prof. Deepak Dhar have mentored me through most of it out of the many academic members at IISER Pune who have assisted me. Dr M.S. Madhusudhan has supported me through difficult times like an older brother.

My parents and sister have always been my pillars of support. They have been incredibly patient and readily available, and I appreciate it. They, each, have made sacrifices for my sake that complete me.

Adarsh, Akhila, Onkar, and Mansi have accompanied me in many of my shenanigans. I have troubled most of them throughout the thesis duration. Aagam, Harshavardhan, Adithya Shetty, and Aditya Banerji have kept me company through incredible fun times. Sanjana, Shomik, Simran, Rahul, Palash, and Prasanna gave me something to look forward to during the bleak times of the pandemic and beyond. Aanjaneya and Neeraj have got me through some difficult times. Vishesh "Capy" Jain has always kept me encouragingly grounded and has given me company. Lastly, Nikita, who has saved me from difficult times, is the person without whom this thesis would've been incomplete.

I want to thank all of them for being there for me.

Abstract

In this thesis, we study the theory of parametrized complexity and examine parameterized complexity of the Hiding Leader problem. We discuss standard tools and techniques for showing the fixed-parameter tractability of parameterized problems like kernelization and branching. We review fixed-parameter intractability and show, using parameterized reductions and W-hierarchy, that some problems are unlikely to be fixed-parameter tractable. Given a graph $G = (V, E)$, a subset $L \subseteq V$ of leaders, an integer k that denotes the maximum number of edges that we are allowed to add in G, an integer d that denotes the least number of followers in $F = V \setminus L$ whose final centrality should be at least as high as any leader, the goal is to compute if there exists a subset $W \subseteq F \times F$ such that (i) $|W| \leq k$, and (ii) there exists a $F' \subseteq F$ with $|F'| \geq d$ satisfying $c(G', f) \geq c(G', \ell)$ for all $f \in F'$ and $\ell \in L$ where $G' = (V, E \cup W)$. We study the parameterized complexity of the problem in the setting of degree centrality for a few sets of parameters and shed light on its fixed-parameter intractability using tools discussed apriori. We obtain the following results for the Hiding Leader problem:

1. The Hiding Leader problem is W[1]-hard when parameterized by $d + k$. Therefore, the problem is W[1]-hard when parameterized by only d or k.

2. The Hiding Leader problem admits a kernel of size $(d - 1)(\Delta + 1) + 1$, where Δ denotes the maximum degree of G.

3. The Hiding Leader problem in the setting of core centrality is $W[1]$-hard when parameterized by $k + d$.

4. Finally, we briefly touch upon the problem and some results in the core-centrality setting.

Contents

Chapter 1

Introduction

In the theory of algorithm design, given a problem, the goal is to find an algorithm that efficiently solves the problem. Although algorithms like Euclid's algorithm or the Egyptian algorithm to multiply numbers have existed since ancient times, the word 'algorithm' is associated closest to the algorithms that run on machines. When we say efficiently, we mean optimally of resources we require for "using" the algorithm (or running the algorithm) to solve a problem. In machines, these resources can be the time taken by the machine to run (time complexity), the storage required by the machine to run the algorithm (space complexity), the energy required by the machine (if any), etc.

Mathematically we usually depict the efficiency of an algorithm using space complexity, time complexity, or both. Moreover, we primarily consider time complexity, expressed as a function of input size (using asymptotic notation). If this function does not increase very much relative to the size of the problem (asymptotically the same as a polynomial function or a slower-growing function), we consider it an efficient algorithm. However, in our search for efficient algorithms, we encounter classes of problems that seemingly do not admit efficient algorithms. One such class of problems is the class of NP-Complete problems. Determining if class P= class NP (P=NP problem) is a long-standing open problem where P is the class of problems that we can solve in polynomial time, and NP is the class of problems whose solutions we can verify in polynomial time.

Given that the P=NP problem is a long-standing open problem, we can assume that no efficient algorithm exists for NP-complete problems (the class of hardest problems in NP).

The natural question follows: Can we do better than exponential running time in the problem size for problems in NP-complete class? If a general problem is NP-complete, we know that an exponential time algorithm will be needed (unless P = NP), but there are a variety of ways which the time complexity of an algorithm can be "exponential" some of which might be preferable to others. The above quote from [12] encapsulates the motivation for the question quite well. In [12], they discuss that it might be preferable to express the time complexity in terms of the parameters that arise naturally out of the problem rather than an artificially constructed problem size. In [21], it was first noted that as k is varied $\Omega(n^{f(k)})$ described the time complexity of problems such as Dominating Set [11]. The database community knew that an intractable problem could turn into a tractable one after fixing a parameter. All these instances lead us to the theory of parameterized complexity consisting of definitions and results in fixed-parameter tractability and intractability.

This thesis will study a few key concepts from the parameterized complexity theory, formally established in [1, 8, 9, 10]. Chapter 1 discusses a few preliminaries required to follow the rest of the thesis. Chapter 2 formally introduces the notion of fixed-parameter tractable algorithms and slice-wise polynomial algorithms. It also introduces the notion of preprocessing called kernelization. Chapter 3 will discuss key techniques to solve a few parameterized problems using branching or bounded search trees. In Chapter 4, we look at some theory developed in [8, 9, 7] to study the intractability of parameterized problems. In chapter 5, we look at the Hiding Leader problem and derive some results in the parameterized setting of the problem, concluding the thesis in Chapter 6

1.1 Original Contributions

This thesis is partly a literature review of the book "Parameterized Algorithms" [4] which introduces key concepts in parameterized complexity. A few of which we use for discussing the tractability and intractability of the Hiding Leader problem discussed in Chapter 5. The thesis gives some new examples for key concepts. Chapter 5 of the thesis discusses new results and new interpretations of existing results for the Hiding Leader problem. We obtain the following results for the Hiding Leader problem:

1. The Hiding Leader problem is W[1]-hard when parameterized by $d + k$. Therefore,

the problem is W[1]-hard when parameterized by only d or k.

2. The Hiding Leader problem admits a kernel of size $(d-1)(\Delta+1)+1$, where Δ denotes the maximum degree of G.

3. The Hiding Leader problem in the setting of core centrality is $W[1]$-hard when parameterized by $k+d$.

1.2 Preliminaries

This section contains a list of definitions and notations from graph theory and algorithms to understand the contents of this thesis.

1.2.1 Graph theory related terminology

For standard notations and definitions in graph theory, we refer to West [23]. We now list several graph definitions being used throughout the thesis.

Definition 1.2.1. A *graph* G is defined as an ordered pair (V, E) where V or $V(G)$ denotes the set of vertices and E or $E(G) \subseteq V \times V$ denotes the set of edges between vertices. Let $u, v \in V$ be two vertices, we denote an edge by (u, v). All of the edges may be directed (from u to v or (u, v)) or undirected (between u and v or (u, v) or (v, u)). A graph with directed edges is called a directed graph and a graph with undirected edges is called an undirected graph.

Definition 1.2.2. A *path* is a sequence of adjacent edges (edges sharing a vertex) with none of the vertices appearing in more than two edges.

Definition 1.2.3. A *directed path* is a path where the direction of the directed edges always points from the previous edge to the next edge.

Definition 1.2.4. A *cycle* is a path starting and ending at the same vertex.

Definition 1.2.5. A *directed cycle* is a cycle where the path is directed.

Definition 1.2.6. A *connected graph* is a special graph where $\forall u, v \in V$ there exists at least one path joining them.

Definition 1.2.7. A *subgraph* $H = (V', E')$ of a graph $G = (V, E)$, written as $H \subseteq G$ is a graph such that $V' \subseteq V \wedge E' \subseteq E$. H is called an vertex-induced subgraph, also written as $G[V']$, if $E' = (V' \times V') \cap E$.

Definition 1.2.8. A *bipartite graph* $G = (V, E)$ with bipartition (A, B) is such that $V = A \cup B$ and $E \subseteq (A \times B)$.

Definition 1.2.9. A *clique* is a set of vertices $C \in V$ of a graph G such that $G[C]$ is a complete graph.

Definition 1.2.10. An *open neighbourhood* of a vertex v, denoted by $N_G(v)$, is the set of all vertices that form an edge with v in G, that is, $N_G(v) = \{u \in V : (u, v) \in E\}$.

Definition 1.2.11. A set of vertices $V' \subseteq V$ of a graph G is called a *connected component* if the induced subgraph $G[V']$ is a connected graph.

Definition 1.2.12. An *independent set* $I \subseteq V$ in the context of graph theory is a set of vertices of a given graph G such that the induced graph $G[I]$ has no edges in it.

Definition 1.2.13. The *degree* of a vertex, denoted by $d_G(v)$, is the number of vertices in its open neighborhood, that is, $d_G(v) = |N_G(v)|$.

1.2.2 Algorithm and complexity related terminology

For standard notations and definitions in algorithms, we refer to Cormen et al. [3]. We now list several definitions being used throughout the thesis.

Definition 1.2.14. An *algorithm* A is a procedure or a set of instructions that takes an input and returns an output, solving a computational problem.

Definition 1.2.15. The *Big-O* notation, written as $O(h(x))$, is a notation used to describe the asymptotic behavior of of a function. Let $f(x), g(x)$ be two functions, we say $f(x) = O(g(x))$ if and only if there exist constants N, C such that $|f(x)| \leq C|g(x)|$ for all $x > N$.

Here is a list of commonly known classes of functions and their names. Here n is a variable and k is a constant real number.

Notation	Name
$O(1)$	constant
$O(log(n))$	logarithmic
$O(n)$	linear
$O(n^2)$	quadratic
$O(n^k)$	polynomial
$O(k^n)$	exponential

Definition 1.2.16. An *universe* Σ^* is a collection of all possible sequences that can be formed using the characters from alphabet Σ (usually $\Sigma = \{0, 1\}$ or some finite set). A *language* L is a subset of the universe, that is, $L \subseteq \Sigma^*$.

Definition 1.2.17. Given a language L and a string $x \in \Sigma^*$, the decision problem is determining if $x \in L$ is true or not.

Definition 1.2.18. A *verification algorithm* $A(x, y)$ is a two-argument algorithm that takes in an input string x and an associated certificate y that verifies x such that $A(x, y) = 1$. If A verifies all strings of a language L, then we say A verifies L.

Definition 1.2.19. The complexity class NP is the set of all languages L (or problems) that a verification algorithm A verifies in polynomial-time such that for all $x \in L$ there exists certificate y such that y is polynomial in $|x|$ and $A(x, y) = 1$.

Definition 1.2.20. We say that there exists a *poynomial-time reduction* from a language L_1 to language L_2 written as $L_1 \leq_P L_2$ where L_1 & $L_2 \in \{0, 1\}^*$ if there exists a computable function f that maps a string x to $f(x)$ in polynomial-time such that $f(x) \in L_2$ if and only if $x \in L_1$.

Definition 1.2.21. A language L is called *NP-hard* if there exists a polynomial-time reduction from every language in NP to L. That is, $L' \leq_P L$ for all $L' \in NP$.

Definition 1.2.22. If a language L belongs to NP and is $NP-hard$ then it is *NP-complete*.

Chapter 2

Parameterized problems and Kernelization

This chapter discusses a few definitions, kernelization reduction rules, and examples. Let us motivate the need to define a parameterized problem through a plausible real-world example.

Suppose we are part of the marketing team for a product. We are to give at most k free samples of the product to some people in some communities to promote the product in those communities. Let us suppose that we know the inner workings of the communities, and we figure out that if all the friends of a particular person in a community have a product, then they get that product, too, out of Fear Of Missing Out (FOMO). Given that we know all the people and their friends in the community, we want to devise a plan to distribute the free samples to some people so that it makes everyone else in the community buy the product. We want to figure out for what communities such a marketing plan is achievable. If achievable, we want to know how to effectively distribute the k free samples. There is little time to develop a plan as the product launch is soon. We start working on the problem and soon realize that for each pair of friends, if both of them do not get a free sample, then neither of them would buy the product. Hence in each pair of friends, at least one friend has to get a free sample for our marketing plan to succeed. We can represent the people of a community using vertices and join the vertices representing two people using an edge if they are friends. We can quickly see that to develop a marketing plan for our requirements; we need to have at least one vertex from every edge in our set of k vertices getting free

samples, that is, we essentially need to fetch a k-sized vertex cover of the graph representing the community. That does not spell profit for the product as this problem is NP-Complete (refer to Definition 1.2.22). That means, unless P=NP, we do not have a polynomial-time algorithm to devise a plan. Even for a community as small as 50 people and $k = 20$ for finding the solution using a brute-force algorithm, it would take the fastest supercomputer at least 300 million years.

However, that does not mean we resign from our job. Notice that if an edge (u, v) of the graph of a community does not get covered in our plan for that community as we are building it, we can branch the plan into two plans, one where u gets a free sample and one where v gets a free sample. This gives us a new way to come up with a plan. Pick an edge where neither u nor v receives a free sample and make two plans, one where u gets a free sample and one where v gets a free sample. For each of these two plans, choose an edge not covered by the plan and create two more plans similarly in each case. This gives us a tree with depth at most k (we cannot give more than k free samples), each internal vertex having two children. The path from the root to a leaf uniquely determines a set of k people getting free samples. There are in total 2^k such paths. If there exists a plan that satisfies our requirements, then it must exist in these 2^k paths. Else a plan meeting our requirements is not feasible. For each path, we see if the set of k vertices obtained is a vertex cover of the graph. This algorithm takes $2^k|E|$ that is at most $2^k n^2$ where n is the number of people in the community. This takes at most 2.7×10^9 calculations which we can write code for on an average everyday computer, and the code will finish computing in a matter of seconds.

This gives us the motivation to study NP-hard problems with some intrinsic parameters to come up with algorithms that may be exponential in the size of the parameter(s), but it is otherwise polynomial in the size of the problem.

2.1 Some formal definitions

For standard notations and definitions in parameterized algorithms, we refer to Cygan et al. [4]. We now list several definitions being used throughout the thesis.

Definition 2.1.1. A *parameterized problem* is a collection of pairs of the form (x, k) where x is a string from the universe Σ^* and k is the parameter associated with the instance of

the problem leaving out the parameter is encoded by $x.$, in other words, it is a language $L \subseteq \Sigma^* \times \mathbb{N}$ (refer Definition 1.2.16).

For example, an instance of INDEPENDENT SET parameterized by k, the size of the independent set (refer Definition 1.2.12) is a pair $(G, k) \in \Sigma^* \times \mathbb{N}$ where G is an undirected graph (refer Definition 1.2.1) that is written as a string over Σ using an appropriate encoding. Note here G is used to denote the graph and the string representing the graph. For (G, k) to belong to INDEPENDENT SET parameterized language, the graph G (or encoded by G) must have an independent set of size k.

Given an instance (x, k) of a parameterized problem, its size is given by $|x| + k$. We now define a specific complexity class to which a parameterized problem may belong.

Definition 2.1.2. A parameterized problem $L \subseteq \Sigma^* \times \mathbb{N}$ is called *fixed-parameter tractable* (FPT) if there exists an algorithm α(called a fixed-parameter algorithm), a computable function $f : \mathbb{N} \to \mathbb{N}$ and a constant c such for all $(x, k) \in \Sigma^* \times \mathbb{N}$, the algorithm α determines whether $(x, k) \in L$ in time bounded by $f(k).|(x, k)|^c$. The complexity class containing all fixed-parameter tractable problems is called FPT.

Sometimes a problem can be parameterized by multiple parameters, say $k, d....$ One can even parameterize the problem by combining two or more parameters (say, k and d). Formally parameterization by k and d is expressed using the parameter $k + d$. It is simple to expand the concept of a parameterized problem and the definition of the FPT class to include more than one parameter. We define k as a vector of non-negative integers, say, of size c (c is a constant), and then the functions f and g can depend on the c parameters encoded by the coordinates of the vector.

2.2 Kernalization

Almost every realistic computer system that seeks to solve an NP-hard problem employs preprocessing (also known as data reduction or kernelization). Preprocessing subroutines aim to efficiently tackle the "easy parts" of a problem instance and condense it down to its ostensibly challenging "core" structure (the kernel of the instance). In other words, this method aims to convert the provided problem instance into a comparable "lower sized"

instance in time polynomially bounded by the input size without necessarily solving the problem. On this smaller instance, one can then apply a slower precise algorithm.

Assume that a suitable preprocessing algorithm substitutes an instance I with an equivalent instance that is at least one bit smaller and runs in polynomial time. It is doubtful that one can come up with such an algorithm as that would mean P = NP. Let such an algorithm exist for an NP-hard decision problem (refer Definitions 1.2.17 and 1.2.21). That would mean that we can run the algorithm, for instance, x of an NP-hard decision problem $|x| - 1$ number of times to reduce the instance x to a single bit instance hence solving the solution in polynomial time. This diminishes the necessity for defining a functional preprocessing approach. However, we can define useful preprocessing in the language of parameterized complexity by stating that huge instances with a small parameter must be downsized. In contrast, small instances relative to their parameter need not be processed further. These form the crux of kernelization.

2.2.1 Formal Definitions

Definition 2.2.1. [4] A *reduction rule* is a function takes an instance (x, k) of a parameterized problem($P \subseteq \Sigma^* \times \mathbb{N}$) and gives an instance (x', k') of the same parameterized problem such that

1 the instance (x', k') is output in polynomial time of the size of (x, k)

2 $(x, k) \in P \iff (x', k') \in P$.

A reduction rule is considered safe if 2 is followed.

Given a pre-processing algorithm for a parameterized problem α, we define the size of the algorithm in terms of the parameter of a parameterized problem.

$size_\alpha(k) = \sup\{|x'| + k' : \alpha(x, k) = (x', k'), x \in \Sigma^*\}$

Definition 2.2.2. [4] We say that a pre-processing algorithm α is a kernelization algorithm or a kernel for a parameterized problem P if α given instance (x, k) outputs (x', k') according to Definition 2.2.1 and $size_\alpha(k) \leq g(k)$ for some computable function g outputting a natural number.

For a kernel we also include 1/0 as outputs (say in the form of $(1, k')/(0, k')$ for an instance (x, k)). This is because, many times, it can happen in the process of kernelization that the instance becomes a trivial yes or no instance.

The primary indicator of a kernelization algorithm's efficiency is a bound on the size of its output. However, in computer science, we typically use an algorithm's running time as an indicator. Though a kernelization technique's actual running time is crucial for practical applications, in theory, a kernelization algorithm simply has to execute in polynomial time.

Clearly, if a problem admits a kernel, given any algorithm that can decide if an instance belongs to the problem or not, the problem becomes FPT. Surprisingly, the converse also holds.

2.2.2 Some simple kernels

Example: Vertex Cover

In the example given at the beginning of this chapter, we have already seen that vertex cover parameterized by solution size k is an FPT problem. So it follows that it must have a kernel for the problem. To achieve this kernel we define a series of successive reduction rules. An instance of the problem is (x, k) where x is a graph $G(V, E)$.

Vertex Cover
Given a graph G the problem is to find set $\|VC\| \leq k$ such that $V - VC$ is an independent set. (refer Definition 1.2.12)

RR VC 1: Remove the set of all isolated vertices (vertices of 0 degree) I from V to define the new instance $(G[V - I], k)$.

This reduction is safe as a vertex from I cannot be part of the vertex cover.

Notice that for a vertex u with degree at least $k + 1$, for each edge, that it is part of, either u has to be in the vertex cover or its corresponding neighbor has to be in the vertex cover. So if u is not in the vertex cover, we cannot have a vertex cover of size at most k. This gives us a new reduction rule.

RR VC 2: If there exists a vertex u such that $d_G(u) > k$, if $k-1 = 0$ and $E(G[V-\{u\}]) \neq \phi$ output $(0, 0)$ (trivially no). If $E(G[V - \{u\}]) = \phi$ while $k - 1 \geq 0$ output $(1, k - 1)$, else output instance $(G[V - \{u\}], k - 1)$,

This reduction is safe due to reasons discussed in the previous argument.

After the application of RR VC 2, we can apply RR VC 1 again and repeat this process until RR VC 2 is no longer applicable or we get a no/yes instance. If RR VC 2 is no longer applicable, then maximum degree of the graph (Δ) of the output instance is k. In case where RR VC 2 is no longer applicable, a k vertex cover can cover at most k^2 edges so the k vertex cover can have at most k^2 neighbors. Therefore for a yes instance of such a kind, we must have that $V(G) \leq k^2 + k$ and $E(G) \leq k^2$. This gives us a polynomial kernel of the Vertex Cover problem where $g(k) = 2k^2$ (refer Definition 2.2.2). Note that all the reductions can be run in $O(V)$ or linear time.

Example: Feedback Arc Set in Tournaments

A tournament is a complete graph T such that for every for every pair of vertices either there is a directed edge from u to v or a directed edge from v to u. A directed edge is called an arc. A *feedback arc set* F is a set of edges for a directed graph G such that $G(V, E - F)$ is a directed acyclic graph, that is, it contains no directed cycle. We note that such an F must contain at least one edge from every directed cycle.

Feedback Arc Set in Tournament
Given a Tournament T the problem is to find set $\|FAST\| \leq k$ such that $T(V, E - FAST)$ is a directed acyclic graph. Or equivalently reversing the edge directions in $FAST$ results in an acyclic tournament.

Similar to RR VC 2, we note that any edge that is part of $k + 1$ directed triangles (directed cycle of length 3) must be part of $FAST$.

RR FAST 1: If there exists (u, v) in $E(T)$ such that it is part of more than k directed triangles, then $E' = E(T) \cup \{(v, u)\} \setminus \{(u, v)\}$, the new instance is $(T = (V, E'), k - 1)$.

The equivalent rule of RR VC 1 is that a vertex u not be in any cycle. Let u in a Tournament such that it is not part of any triangles in T. Consider two sets, $v^+ = \{u :$

$(v,u) \in E(T)\}$ and $v^- = \{u : (u,v) \in E(T)\}$, $V = v^+ \cup v^- \cup \{v\}$. We cannot have a cycle contained in either $v^+ \cup \{v\}$ or $v^- \cup \{v\}$ that also contains v in it. So a cycle in T containing v must have an edge from v^+ to v^-, resulting in a triangle containing v, but since v is not part of any triangles, there exists no such edge. This implies that v is not part of any cycles. This also implies that cycles in T are either contained in v^+ or v^-. $FAST(T) = FAST(T[v^+]) \cup FAST(T[v^-])$. Hence removal of v from $V[T]$ is safe.

RR FAST 2: If there exists a vertex v which is not part of any triangle, delete v from $V[T]$.

After applying RR FAST 1 and 2 the output graph is still a tournament. Let us apply them until both of them are no longer applicable. If the resulting instance is a yes-instance, then we have $|FAST| \leq k$. Since the graph is still a tournament, $V[FAST] \cup N_G(V[FAST]) = V$, every vertex is either part of $FAST$ or forms a triangle with some edge in $FAST$, but the k arcs each can be part of at most k triangles. Hence $|V[T]| \leq k(k+2)$. Hence we have a polynomial kernel for Feedback arc set in tournaments with $g(k) = 3k^2$.

2.2.3 Crown Decomposition

One of the broad kernelization methods we can use to find kernels for various problems is crown decomposition. The method is based on the well-known König and Hall matching theorems.

A matching M in a graph G is a set of disjoint edges, that is, no two edges from M share a vertex.

Definition 2.2.3. A partitioning of $V(G)$ into C, H and R is called a *crown decomposition* if

1. $C \neq \phi$
2. C is an independent set
3. $((C \times R) \cup (R \times C)) \cap E(G) = \phi$ or H is a separator for C and R
4. Define $E' = ((C \times H) \cup (H \times C)) \cap E(G)$, there exists a matching $M \subseteq E'$, such that $|M| = |H|$, that saturates H. That is all the vertices of H are matched by M.

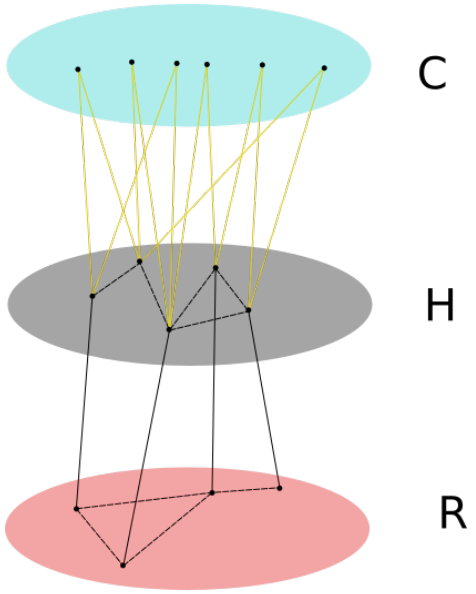

Figure 2.1: Example of a Crown decomposition

It's as if we have placed a crown C on the head H of a person, with R being the rest of her body. Figure 2.1 shows such a decomposition.

Theorem 2.2.1. ***König's Theorem*** *[18, 4] "In every undirected bipartite graph, the size of a maximum matching is equal to the size of a minimum vertex cover"*

Theorem 2.2.2. ***Hall's theorem*** *[14, 4] "Let G be an undirected bipartite graph with bipartition (V_1, V_2). The graph G has a matching saturating V_1 if and only if for all $X \subseteq V_1$, we have $|N(X)| \geq |X|$" (refer Definition 1.2.8).*

Theorem 2.2.3. ***Hopcroft-Karp A lgorithm*** *[17, 4] "Let G be an undirected bipartite graph with bipartition V_1 and V_2, on n vertices and m edges. Then we can find a maximum matching and a minimum vertex cover of G in time $O(m\sqrt{n})$. Furthermore, in time $O(m\sqrt{n})$ either we can find a matching saturating $V_!$ or an inclusion-wise minimal set $X \subseteq V_1$ such that $|N(X)| < |X|$" (refer Definition 1.2.8).*

We will use Theorem 2.2.1, 2.2.2, 2.2.3 to show the following.

Lemma 2.2.4. *Let there be a graph G with no vertices of degree 0 such that $|V(G)| \geq 3k+1$. Then there exists an algorithm α running in $O(|G|^c)$ where c is a constant such that:*

- *α produces a matching $|M| = k + 1$ or*

- *α produces a crown decomposition of G.*

Proof. We first use a greedy algorithm to find a matching M in G such that there exists no matching M' of G exists such that $M \subset M'$. If $|M| > k$, stop the algorithm. Else size of M is at most k. Let A be the set such that $uv \in M \implies u, v \in A$. The size of A is at most $2k$. If $G[B = V(G) \setminus A]$ has an edge, we can include it in M to produce M' such that $M \subset M'$, but since that is not possible, B is an independent set.

Consider a bipartite graph $G_{B,A}$, which contains only edges between B and A in G. We can use 2.2.3 to construct a maximum matching M' and a minimum vertex cover VC of $G_{B,A}$. As before, if M' is of size greater than k, then stop the algorithm. Else M' is of size at most k. We have by 2.2.1 that M' and VC are of the same size $\implies$ VC is of at most size k.

Let's say that $VC \cap A = \phi$, this means $VC \subseteq B$. Notice that there exists no $u \in B \setminus VC$ as if such a u exists, it must have an edge in $G_{B,A}$ (no vertices with 0 degree in G and u has no edges in $G[B]$) but since VC is vertex cover of $G_{B,A}$ it must contain one of the endpoints of that edge. This would imply $u \in VC$ as the other endpoint is in A and $VC \cap A = \phi$ which contradicts $u \in B \setminus VC$. Hence $VC \cap A = \phi \implies |B| = |VC| \leq k$. Then $|B| + |A| \leq 3k + 1$ but since $|V(G)| \geq 3k + 1 \implies VC \cap A = \phi$ is false.

Therefore VC intersects with the set A. Define a matching M^* as the matching that saturates $VC \cap A$ and $A*$ as the set of endpoints of M^*. Each edge of M^* contains exactly one endpoint in $VC \cap A$. $H = VC \cap A = VC \cap A*$, $C = A^* \cap B$ and $R = V(G) \setminus (C \cup H) = V(G) \setminus A^*$ is the required crown decomposition. M^* saturates $VC \cap A^* = H$, since B is independent set $C \subseteq B$ is also independent. Since vertices from $C = B \cap A^*$ can only have edges in M^* and all endpoints of M^* = vertices of $A*$ are removed from R there can be no edges from R to C. Hence H is a separator of C and R. $\square$

2.2.4 Application to Vertex Cover

We use the previously defined RR VC 1 to remove isolated vertices in a given instance (G, k). If $|V(G)| \geq 3k + 1$, according to Lemma 2.2.4, we have that either a matching of $k + 1$ size exists which cannot be covered by a vertex cover of size k, conclude it is a no instance.

Else we have a crown decomposition $C \cup H \cup R$ where a matching M exists that saturates H into C. Notice that if we don't take H into the solution set of the given instance, we need at least $|H|$ vertices from C to cover the edges of M, and since H also covers all incident edges from R to $C \cup H$ the solution size is minimized if we take vertices of H. The new instance is $(G[V - V(H)], k - |H|)$. We can keep applying this reduction again after applying RR VC 1 to obtain a $H' \neq \phi$ and consequently

$$(G[V \setminus V(H) \cup V(H')], k - |H| - |H'|).$$

We can repeat this as long as $|G| \geq 3k + 1$. Therefore the Vertex Cover problem admits a kernel with at most $3k$ vertices.

2.2.5 Kernel for Vertex Cover Problem based on linear programming

We will now see how to arrive at a $2k$ kernel for Vertex Cover using Linear programming. Let us encode a Vertex Cover instance as an Integer Linear Programming instance. For each vertex $v \in V(G)$ of a Vertex Cover instance, we introduce a variable x_v. If a vertex v is to be included in the vertex cover, then x_v is set to 1. Otherwise, x_v is set to 0. Each $(u, v) \in E(G)$ has to be covered by at least one vertex. So $x_v + x_u \geq 1$ for every $uv \in E(G)$. We need to minimize $\sum_{v \in V(G)} x_v$. That concludes the ILP formulation.

But this problem is at least as hard as k-Vertex Cover problem as if $\sum_{v \in V(G)} x_v \leq k$ we have a vertex cover of size at most k. To make the problem easier, we relax the condition that all x_v are integers, allowing x_v to take real values. We call this relaxation LPVC(G). LPVC(G) does not necessarily encode the Vertex Cover problem but its optimum solution is useful for us. LPVC(G) can be solved in polynomial time since it is an instance of Linear Programming.

For an optimal solution of LPVC(G) consider a partition of the variables $x_v \in [0,1]$ as such.

- $V_0 = \{v \in V(G) : x_v < \frac{1}{2}\}$
- $V_{\frac{1}{2}} = \{v \in V(G) : x_v = \frac{1}{2}\}$
- $V_1 = \{v \in V(G) : x_v > \frac{1}{2}\}$

Theorem 2.2.5. ***(Nemhauser-Trotter theorem)****[4] There is a minimum vertex cover S of G such that $V_1 \subseteq S \subseteq V_1 \cup V_{\frac{1}{2}}$*

Using Theorem 2.2.5 we can define the following reduction.

RR VC 3: Consider an optimum solution to LPVC(G) and partitions V_0,$V_{\frac{1}{2}}$ and V_1 as defined above. If $\sum_{v \in V(G)} x_v > k$ conclude G is a no-instance. Else take vertices of V_1 into the vertex cover, decrease k by $|V_1|$ and delete all vertices of $V_1 \cup V_0$.

Note that after application of **RR VC 3** we are left with $V_{\frac{1}{2}}$ such that $|V_{\frac{1}{2}}| \leq 2k$. This gives us a $2k$ kernel for vertex cover.

Chapter 3

Bounded Search Trees

This chapter discusses the bounded search tree approach, a variation of exhaustive search, one of the most often employed tools in the design of fixed-parameter algorithms. We use methods for parameterization of the Vertex Cover problem to demonstrate this technique.

One of the simplest and most popular parameterized complexity algorithms, *bounded search trees*, or simply *branching*, is based on the concept of backtracking. By making a series of decisions on the problem's structure, such as whether or not to include a particular vertex in the solution, the algorithm attempts to construct a workable solution. The algorithm branches into several subproblems that are solved one at a time while considering one of these steps, looking into many options for the decision. In this way, the execution of a branching algorithm can be compared to a *search tree* that the algorithm traverses until it finds the solution in one of the leaves. We must contend that in the case of a yes instance, some decisions captured by the algorithm lead to a workable solution to defend the validity of a branching algorithm. Such a branching algorithm works in FPT time if the total size of the search tree is constrained by a function of the parameter alone and every step requires polynomial time. A lot of natural backtracking algorithms share this feature.

Let X represent a specific instance of a minimization issue, like Vertex Cover. When using FPT algorithms, the measure $\mu(X)$ we associate with the instance X is often a function of k by itself. We create simpler versions of the same problem from X in a branch phase $(x_1, ..., x_l)$ such that the following hold:

1. Every viable solution $h_i(S)$ of X corresponds to every feasible solution S of $X_i \in \{X_1, ..., X_l\}$. Additionally, there is at least one optimum solution for X in the set $\{h_i(S) : 1 \leq i \leq l$, and S is a possible solution of $X_i\}$. Informally, a branch step divides problem X into subproblems $X_1, ..., X_l$, while potentially making certain (formally justified) avaricious choices.

2. The number l is small; for instance, it is constrained by the function $\mu(X)$ alone.

3. In addition, we have that for every $X_i, i \in \{1, ..., l\}$, $\mu(X_i) \leq \mu(X) - c$ for some constant $c > 0$. In other words, we greatly simplify the current instance in each branch.

A branching algorithm recursively applies branching steps to instances $X_1, X_2, ..., X_l$ until they become simple or trivial. Thus, we can visualize the algorithm's execution as a search tree, with each recursive call corresponding to a node: the calls on instances $X_1, X_2, ..., X_l$ are children of the call on instance X. The second and third conditions allow us to limit the number of nodes in this search tree, assuming that the instances with the non-positive measure are simple. Indeed, the third condition allows us to limit the depth of the search tree in terms of the original instance's measure, whereas the second condition controls the number of branches beneath each node. Because of these characteristics, search trees of this type are frequently referred to as *bounded search trees*. A branching algorithm with a carefully chosen branching step frequently outperforms a simple exhaustive search.

We present a typical scheme for using bounded search trees to design parameterized algorithms. In polynomial time, we first identify a small (typically constant or bounded by a function of the parameter) subset S of elements, at least one of which must be in *some* or *all* feasible solutions to the problem. Then we solve $|S|$ subproblems: for each element e of S, we create one subproblem that includes e in the solution, and we solve the remaining task with a lower parameter value. We also say that we branch on the solution-related element of S. The drop of the parameter in each branch of such search trees is measured. We can bound the depth of the search tree by a function of the parameter if we ensure that the parameter (or some measure bounded by a function of the parameter) decreases by at least a constant value in each branch, resulting in an FPT algorithm.

Often, it is simpler to think of branching as "guessing" the correct branch. That is, whenever a branching step is performed, the algorithm attempts all possibilities to guess the correct part of an (unknown) solution in the graph. We must ensure that a series of guesses uncovers the entire solution and that the total time spent on incorrect guesses is reasonable.

3.1 Vertex Cover

The strategy is used on Vertex Cover as an example of branching. In Chapter 2, we presented a kernelization algorithm that constructs a kernel on at most $2k$ vertices in time $O(\sqrt{m}n)$. To solve Vertex Cover in time of $O(n\sqrt{m} + 4^k k^{O(1)})$, kernelization can be easily combined with a brute-force algorithm. In fact, a 2k-vertex graph has no more than two $2^{2k} = 4^k$ subsets of size at most k. Thus, we can solve the problem in time $O(n\sqrt{m} + 4^k k^{O(1))}$ by enumerating all vertex subsets of size at most k in the kernel and checking whether any of these subsets forms a vertex cover. By branching, we can easily obtain a better algorithm. This algorithm was introduced in Chapter 2 under the guise of coming up with a marketing strategy of distributing free samples of a product to communities.

Assume that (G, k) is a Vertex Cover instance. Our algorithm is founded on two simple observations.

- Any vertex cover for a vertex v must contain either v or all of its neighbors $N_G(v)$.
- When the maximum degree of a graph is at most 1, Vertex Cover becomes trivial (it can be solved optimally in polynomial time)

We will now go over our recursive branching algorithm. Given an instance (G, k), we first find the maximum degree vertex $v \in V(G)$ in G. If all v have degree 1, then each connected component of G is an isolated vertex or edge, and the instance has a trivial solution. Otherwise, $|N(v)| \leq 2$ and we recursively branch on two cases in the vertex cover by considering either v or $N_G(v)$. We can delete v and reduce the parameter by one in the branch where v is in the vertex cover. The second branch adds $N_G(v)$ to the vertex cover, deletes $N_G[v]$ from the graph, and reduces k by $|N_G(v)| \leq 2$. The algorithm's running time is limited by (the number of nodes in the search tree) $\times$ (time taken at each node).

Clearly, the time spent at each node is constrained by $n^{O(1)}$. Thus, if $\tau(k)$ is the number of nodes in the search tree, the algorithm's total time is at most $\tau(k)n^{O(1)}$.

Every internal node of $\mathcal{T}$ has at least two children in every search tree $\mathcal{T}$ that corresponds to a run of a branching algorithm. As a result, if $\mathcal{T}$ has l leaves, the number of nodes in the search tree is at most $2l-1$. As a result, limiting the number of leaves in the corresponding search tree is sufficient to limit the running time of a branching algorithm.

In our case, the tree $\mathcal{T}$ represents the algorithm's search tree when run with parameter k. It has two subtrees beneath its root: one for the same algorithm run with parameter $k-1$ and one for a recursive call with parameter at most $k-2$. The same pattern can be found deeper in $\mathcal{T}$. This means that if we define a function $T(k)$ using the recursive formula

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

the number of leaves of $\mathcal{T}$ is constrained by $T(k)$.

We show that $T(k)$ is bounded by 1.6181^k using induction on k. This is true for $k = 0$ and $k = 1$, so let us move on to $k \geq 2$:

$$\begin{aligned} T(k) = T(k-1) + T(k-2) \;&\leq\; 1.6181^{k-1} + 1.6181^{k-2} \;\leq\; 1.6181^{k-2}(1.6181+1) \\ &\leq\; 1.6181^{k-2} \;\leq\; 1.6181^2 \;\leq\; 1.6181^k \end{aligned}$$

This demonstrates that the number of leaves is limited to 1.6181^k. Combined with kernelization, we obtain an algorithm that solves Vertex Cover in time $O(n\sqrt{m} + 1.6181^k k^{O(1)})$.

The obvious question is how we knew that 1.6181^k is a solution to the preceding recurrence. Assume we want to find an upper bound on the function $T(k)$ of the form $T(k) \leq c \cdot \lambda^k$, where $c > 0, \lambda > 1$ are some constants. We can set constant c to satisfy the initial conditions in the definition of $T(k)$. Then we must use induction to demonstrate that this bound holds for all k. This boils down to proving that

$$c \cdot \lambda^k \geq c \cdot \lambda^{k-1} \geq c \cdot \lambda^{k-2} \tag{3.1}$$

because then we will have

$$T(k) = T(k-1) + T(k-2) \ \leq c \cdot \lambda^k \geq c \cdot \lambda^{k-1} \geq c \cdot \lambda^{k-2}.$$

Because Equation (3.1) is equivalent to $\lambda^2 \geq \lambda + 1$, it makes sense to look for the lowest possible value of λ for which this inequality is satisfied; this is the one for which equality holds. By solving equation $\lambda^2 = \lambda + 1$ for $\lambda > 1$, we find that $\lambda = \frac{1+\sqrt{5}}{2} < 1.6181$, so the inductive proof works for this value of λ.

The above algorithm's running time can be easily improved using the following argument:

Proposition 3.1.1. *When the maximum degree of a graph is at most 2,* Vertex Cover *can be solved optimally in polynomial time.*

As a result, we only branch on vertices of at least degree 3, which immediately leads us to the following upper bound on the number of leaves in a search tree:

$$T(k) = \begin{cases} T(k1) + T(k3) & \text{if } k \geq 3, \\ 1 & \text{otherwise.} \end{cases}$$

Again, for the above recursive function, an upper bound of the form $c \cdot \lambda^k$ can be obtained by finding the largest root of the polynomial equation $\lambda^3 = \lambda^2 + 1$. The root is estimated to be at most 1.4656 using standard mathematical techniques (and/or symbolic algebra packages). When combined with kernelization, we get the following theorem.

Theorem 3.1.2. *[4]* Vertex Cover *has a time complexity of* $O(n\sqrt{m} + 1.4656^k k^{O(1)})$.

Can we use a similar strategy for graphs with minimum degree four? This becomes more difficult because Vertex Cover is NP-hard on graph with degree at most three. However, there are more involved branching strategies and faster branching algorithms than the one presented in Theorem 3.1.2.

3.2 How to Handle Recursive Relationships

We must limit the number of nodes in the search tree to obtain an upper bound on the running time of an algorithm based on the bounded search tree technique. Recurrence relations are used for this. We have the most common case in parameterized branching algorithms when we use linear recurrences with constant coefficients. For this case, there is a standard technique for limiting the number of nodes in the search tree. Suppose the algorithm solves a problem of size n with parameter k and recursively calls itself on problems with decreased parameters. In that case, $k - d_1, k - d_2, ..., k - d_p$, then $(d_1, d_2, ..., d_p)$ is called the *branching vector* of this recursion. For example, in the previous section, we used a branching vector (1, 2) to obtain the first VERTEX COVER algorithm and a branching vector (1, 3) to obtain the second. For a branching vector $(d_1, d_2, ..., d_p)$, the following linear recurrence gives information on the upper bound $T(k)$ on the number of leaves in the search tree:

$$T(k) = T(k - d_1) + T(k - d_2) + ... + T(k - d_p).$$

Again, we set the initial condition $T(k) = 1$ for $k < d$, where $d = max_{i=1,2,...,p} d_i$. Assuming that new subproblems with smaller parameters can be solved in polynomial time in n, the running time of such a recursive algorithm is $T(k) \cdot n^{O(1)}$.

If we now seek an upper bound of the form $T(k) \leq c \cdot \lambda^k$, the inductive step is reduced to proving the following inequality:

$$\lambda^k \geq \lambda^{k-d_1} + \lambda^{k-d_2} + ... + \lambda^{k-d_p}$$

The above inequality can be expressed as $P(\lambda) \geq 0$, where

$$P(\lambda) = \lambda^d - \lambda^{d-d_1} - \lambda^{d-d_2} - ... - \lambda^{d-d_p}$$

is the characteristic polynomial of the recurrence for $T(k)$ (recall that $\mathrm{d} = d = max_{i=1,2,...,p} d_i$). It is not difficult to demonstrate using standard calculus techniques that if a polynomial P has the form shown above, then P has a unique positive root λ_0 and $P(\lambda) < 0$ for $0 < \lambda < \lambda_0$ and $P(\lambda) > 0$ for $\lambda > \lambda_0$. This means that λ_0 is the best possible value for an upper bound for $T(k)$.

The *branching number* corresponding to the branching vector$(d_1, d_2, ..., d_p)$ is frequently referred to as the root λ_0. As a result, the branching algorithm's running time is limited to $\lambda_0^k n^{O(1)}$.

Two obvious questions arise:

- How accurate is $T(k)$ estimation using the exponent of the corresponding branching number?
- How well does $T(k)$ predict the size of the search tree?

The answer to the first question is "it is good": the estimation is accurate up to a polynomial factor. The second question is much more difficult because the way a branching procedure explores the search space may be more complex than our recursive formula-based estimation of its behavior. If, for example, a branching algorithm employs multiple methods of branching into subproblems (so-called branching rules) that correspond to different branching vectors and/or is combined with local reduction rules, we do not yet know how to estimate the running time better than by using the branching number corresponding to the worst branching vector. However, the delicate interplay of different branching rules and reduction rules may result in a much smaller tree than our imprecise estimates.

Chapter 4

Fixed-Parameter Intractability

In this chapter, we examine the related issues surrounding lower bounds, namely how to demonstrate (provide proof) the improbability of a fixed-parameter tractable algorithm for a parameterized problem. This chapter establishes a lower-bound theory for parameterized problems analogous to the polynomial-time computation NP-completeness theory. We assume the practical perspective of the algorithm designer, offering evidence for the absence of algorithms that meet a given set of requirements for as many situations as possible.

We cannot rule out the idea that issues like Clique and Dominating Set are polynomially solvable and thus FPT because we do not have proof that P $\neq$ NP. To prove assertions of the form "if problem A has a specific type of algorithm, then problem B has a certain type of algorithm as well," our lower limit theory must be conditional. If it is agreed upon as a working hypothesis that problem B lacks such algorithms, then it follows that problem A also lacks such an algorithm. We need a concept of reduction that transfers (negative evidence for) fixed-parameter tractability from one problem to another to verify such statements in the setting of fixed-parameter tractability.

Reductions from Clique offer tangible proof that some parameterized problems are not in FPT if we take the working hypothesis that Clique itself is not in FPT. Although P $\neq$ NP is a strong assumption, we need one that is even stronger (Clique is not FPT) because we do not yet know how to base fixed-parameter intractability results just on P $\neq$ NP.

4.1 Paremeterized Reductions

Recall the concept of polynomial-time reduction (refer to definition 1.2.20). It gave us the machinery required to prove that a problem A is just as hard as problem B, a polynomial time reduction from A to B would imply that if there exists a polynomial running time algorithm for B, then there would exist a polynomial time algorithm for A.We require similar machinery for parameterized problems that can transfer the membership to the FPT Class of problems.

Definition 4.1.1. [4] For two parameterized problems L_1 and L_2 we define an algorithm α that maps instances of L_1 $((x,k))$ to equivalent instances of L_2 $((x',k'))$. The algorithm α is said to be a *parameterized reduction* if:

1. $(x,k) \in L_1 \iff (x',k') \in L_2$
2. k' is bounded by a computable function, $g(k)$
3. α runs in FPT time of (x,k).

Drawing parallels to polynomial-time reductions, we want to show the transference of membership in FPT. We assume that all computable functions f, g are non-decreasing as given any f, g, we can construct non-decreasing versions of them.

Theorem 4.1.1. *Let α be a parameterized reduction from L to L'. If $L' \in FPT \implies L \in FPT$*

Proof. For every instance (x,k) of L we have $\alpha((x,k)) = (x',k')$ where $k' \leq g(k)$ for some commutable g, since α runs in $f(k)|x|^{c_1}$ it cannot output an instance whose size is larger than its running time implies $|x'| \leq f(k)|x|^{c_1}$. Since L' is in FPT, $\exists$ algorithm β which determines if $(x',k) \in L'$ in $h(k')|x'|^{c_2} \leq h(g(k))(f(k)|x|^{c_1})^{c_2}$ and that also determines if $(x,k) \in L$. Hence we have an algorithm that runs in $h(g(k))(f(k)|x|^{c_1})^{c_2} + f(k)|x|^{c_1)} \leq f'(k)|x|^{c_1 c_2}$ that determines if $(x,k) \in L$ for every instance (x,k). Here $f'(k) = f(k) + h(g(k))f(k)^{c_2}$.

4.1.1 Comments on Parameterized Reductions

One of the easiest parameterized reductions is the reduction from the k-Independent Set problem to the k-clique problem. If $\exists$ is an independent set in a graph G, then a clique of the same size in $\overline{G}$ exists. Hence $G \to \overline{G}$ is the required parameterized reduction. Here instance (x, k) of the Independent set problem is mapped to an instance $(\overline{x}, k)$ of the clique problem in polynomial time. Hence, a parameterized reduction exists from k-Independent set to k-Clique problem and vice versa. This shows that k-Independent set and k-clique problem are equally hard.

We notice that not every polynomial-time reduction used in NP-completness proofs translates into a parameterized reduction as the parameter of the equivalent instance isn't required to be bounded by a function of the old parameter in the case of a polynomial-time reduction. We will see a simple example of this case. However, most NP-completeness proofs translate into parameterized reductions.

If we have a k-Vertex cover for (VC) an instance (G, k) then for the same graph $G[V \backslash VC]$ is an independent set of size $|V| - k$. Therefore an instance $(x, k) \in$ Vertex Cover $\iff$ $(x, |x| - k) \in$ Independent set. This is a polynomial time reduction. This, however, is not a parameterized reduction as $|x| - k$ is not bounded by any computable $g(k)$. Since an independent set problem is as hard as the clique problem, and it is our working assumption that clique is not believed to be in FPT, it is unlikely for a parameterized reduction from independent set to Vertex cover to exist, as the latter problem is proven to be in FPT.

4.2 Problems at least as hard as Clique

We will see how parameterized reductions behave under composition.

Theorem 4.2.1. *[4] If there exists α that is a parameterized reduction from L_1 to L_2, and a parameterized reduction β from L_2 to L_3 then there exists parameterized reduction $(\beta(\alpha))$ from L_1 to L_3.*

The above theorem can be proved in a similar fashion to theorem 4.1.1. The pseudo-proof is as follows; We employ the result that α is FPT time in the instance of L_1, and for the

equivalent instance of L_2 produced by α its size is bounded by FPT size of an instance of L_1 where the parameter obtained after applying α is bounded by some non-decreasing function of the parameter of an instance of L_1. This, along with the existence of β, gives us a reduction $\beta(\alpha)$ that is a parameterized reduction from L_1 to L_3.

Theorem 4.2.2. *[4] A parameterized reduction α, from* Clique *to* Clique on Regular Graphs*, exists.*

Proof. Given an instance (G, k) of clique, we construct a new graph G^r. If Δ is the maximum degree of any vertex in G, the construction is as follows:

- Construct Δ copies of G; $G_1, G_2, ..., G_\Delta$ where let $v_i \in V(G_i)$ correspond to $v \in V(G)$
- For a vertex v place a set of vertices $|V_v| = \Delta - d_G(v)$. connect all v_i to all vertices of this set.

Observe that G' is a Δ regular graph. We claim that G has a clique of size k if and only if G' has a clique of size k. In the forward direction, if a clique of size k exists in G , then there are Δ copies of it in G'. In the reverse direction, if a clique of size k exists in G' it cannot contain vertices from V_v for any v as $\forall\ u \in V_v\ \forall\ v,\ N_{G'}(u)$ is independent set. Also, since u_i is not connected to v_j $\forall u, v, i \neq j$. We have that the clique must be contained entirely in some G_i which implies that k-clique exists in G. This proves the claim. Furthermore, the reduction is a polynomial-time reduction. □

Theorem 4.2.3. *There exists α, a parameterized reduction from* Clique *to* Independent Set *on regular graphs.*

This result is a direct consequence of the fact that the complement of a regular graph is regular, so there exists a parameterized reduction from Clique on regular graphs to Independent Set on regular graphs. Using theorem 4.2.1, we get the proof for Theorem 4.2.3.

4.2.1 Other known reductions from Clique

Theorem 4.2.4. *[4] There exist parameterized reductions from* Clique *to the following problems:*

- Partial Vertex Cover
- Multicoloured Clique
- Multicoloured Independent Set
- Dominating Set
- Set Cover
- Dominating Set on Tournament
- Connected Dominating Set

4.3 W-hierarchy

To accurately convey the complexity of various hard parameterized issues, Downey and Fellows [11] developed the W-hierarchy. In this part, we briefly review the key terms and findings pertaining to this hierarchy.

Definition 4.3.1. Weighted Circuit Satisfiability (WCS) Problem [4] Given a circuit C and an integer k, the goal is to find exactly k input nodes which when given input 1 satisfy the circuit.

Definition 4.3.2. W-hierarchy[4] A parameterized problem P belongs to the class $W[t]$ if there is a parameterized reduction from P to $WCS[\mathcal{C}_{t,d}]$ for some $d \geq 1$.

Here $WCS[\mathcal{C}_{t,d}]$ represents the restriction of WCS on the class of circuits $\mathcal{C}_{t,d}$ where d denotes the depth of the circuit and t denotes the weft of the circuit. The weft of a circuit is the maximum number of large nodes encountered in a path from input to output maximized over different paths. A large node is a node with an in-degree not bounded by any constant.

Theorem 4.3.1. *[4]* Independent Set *is* $W[1]$*-complete.*

Consider the circuit expressing an Independent Set with a particular form: one layer of negation nodes, one layer of small or-nodes with in-degree 2, and a final layer consisting of a single large and-node. Theorem 4.3.1 shows that this form is in some ways canonical for weft-1 circuits: the satisfiability problem for (bounded-depth) weft-1 circuits can be reduced to weft-1 circuits of this form.

Theorem 4.3.2. *[4] The* Dominating Set*,* Set Cover*, and* Hitting Set *are all* $W[2]$*-complete.*

The W-hierarchy is commonly assumed to be a proper hierarchy: $W[t] \neq W[t+1]$ for every $t \geq 1$. Based on Theorem 4.3.2, we do not expect a reduction in the opposite direction: this would imply that the Independent Set is both $W[1]$- and $W[2]$-complete, and thus $W[1] = W[2]$.

Theorem 4.3.3. *[4]* $W[1]$*-complete parameterized problems include:*

- Clique
- Multicolored Clique
- Independent Set
- Multicolored Independent Set
- Partial Vertex Cover

Chapter 5

The Hiding Leader Problem

5.1 Introduction

A covert network is a social network which has one or many harmful users. Social Network Analysis (SNA) tries to reduce criminal activities (e.g., counter terrorism) via detecting the influential users in such networks. There are various popular measures to quantify how influential or central any user or vertex is in a network. As expected, strategic and influential miscreants in covert networks would try to hide herself and her partners (called *leaders*) from being detected via these measures by introducing new edges. Waniek et al. [22] first propose the Hiding Leader problem which incorporates the viewpoint of the leaders of a criminal organization. It also explicitly models knowledge of the criminals about SNA tools that are used to detect them and thus help in dismantling their organization. Intuitively, the input in the Hiding Leader problem is a network with a subset of vertices marked as leaders. The goal is to add fewest edges to ensure that various SNA tools do not rank any leader high based on centrality measures.

5.1.1 Centrality measures

Degree Centrality: One of the centrality measures used in social network analysis is the degree of a vertex. That is, $C(G, y) = d_G(y)$. The Hiding Leader problem seems easy

at least when the centrality measure is given by the degree of a vertex. However, it has been shown using a reduction from the NP-complete CLIQUE problem that this problem is NP-hard [22] (refer Definitions 1.2.22 and 1.2.21). We will later use the same reduction used in [22] to arrive at a different hardness result. In this project, we mostly consider the degree centrality measure.

Core Centrality: Let $r \geq 2$ be a fixed integer. The r-core of a graph G is the largest induced subgraph of G in which all vertices have degree at least r.

$$C_{\text{core}}(G, v) = \max\Big\{r : v \text{ belongs to some } r\text{-core of } G\Big\}.$$

5.1.2 Problem Statement

We now give a formal definition of the HIDING LEADER problem.

Definition 5.1.1. Given a graph $G = (V, E)$, a subset $L \subseteq V$ of leaders, an integer k that denotes the maximum number of edges that we are allowed to add in G, an integer d that denotes the least number of followers in $F = V \setminus L$ whose final centrality measure should be at least as high as any leader, the goal is to compute if there exists a subset $W \subseteq F \times F$ such that

1. $|W| \leq k$,
2. there exists a $F' \subseteq F$ with $|F'| \geq d$ satisfying $C(G', f) \geq C(G', \ell)$ for all $f \in F'$ and $\ell \in L$ where $G' = (V, E \cup W)$.

In the above definition $C(.,.)$ denotes either degree centrality or core centrality. The goal is to have at least d vertices such that their centrality measure (given by the function C) is greater than or equal to that of any of the leaders. In social network analysis tools used to analyze covert networks, one of the tools to determine its most influential nodes is to rank the nodes using their centrality measures. Here d is the safety margin.

Since we are considering a social network, deleting an edge could mean the loss of an essential communication line. Hence we avoid removing edges. Since adding edges has the cost of creating fake edges in the network, we want the cost to be bounded. That is, not

more than k edges can be added to hide the leader (influential nodes). To avoid making the leaders more suspicious, we ensure not to increase their degrees (degree is one of the most commonly used centrality measures). Hence we choose the set of edges W to be added from possible edges between the followers ($F \times F$)

We consider four parameterizations of the problem

1. d
2. k
3. $d + k$
4. $\Delta + d$

When we say we parameterize the problem by $d+k$, we consider both d and k as parameters to the problem, that is, for an instance $(x, d+k)$ we try to find an FPT algorithm that runs in $O(f(d,k).|(x,d+k)|^c)$ time, where f is a computable function and c is a constant. Clearly if such an FPT algorithm exist then it is also $O(f(d+k).|(x,d+k)|^c)$. Similarly, when we consider the parameterization $\Delta + d$ we consider both Δ and d as parameters to the Hiding Leader problem.

5.2 An approximation algorithm

We present a very simple algorithm for approximating the allowed budget k up to a factor of two.

Consider an instance (G, L, k, d) of the Hiding Leader problem. Let C_{max} be the highest degree of the leader vertices, that is,

$$C_{max} = \max\{d_G(u) \ : \ u \in L\}.$$

We rank the vertices of $V \setminus L$ with their degree and pick the top d vertices. For this set of d vertices, we satisfy each vertex by adding enough random edges to it and coloring the

edges to distinguish between old and new edges. It is sufficient to make the degrees of these vertices equal to C_{max}

Let the vertices be $S = \{u_1, u_2...u_d\}$. We now count the number of edges added to the graph. For each $u_i \in S$, the number of edges added is at most $C_{max} - d_G(u_i)$. So the total number of edges added is at most

$$\sum_{i=1}^{d}(C_{max} - d_G(u_i)) = d * C_{max} - \sum_{i} d_G(u_i).$$

This gives us an approximation of k up to a factor of 2. Consider degree sum of vertices in S, that is, $d_G(S) = \sum_{i=1}^{d} d_G(u_i)$. The goal is to achieve $d_G(S)$ to at least $d * C_{max}$. Each edge from the budget can contribute in $d_G(S)$ at most 2. Therefore we need at least

$$\frac{d * C_{max} - \sum_{i=1}^{d} d_G(u_i)}{2}$$

edges to hide all the leaders using S. However, $d_G(S)$ is the maximum among degree sums of all d sized subsets of V. Therefore for the instance (G, L, k, d) the minimum budget required is approximated by $d * C_{max} - \sum_{i=1}^{d} d_G(u_i)$ up to a factor of 2.

5.3 W[1]-hardness parameterized by $d + k$

We now give a parameterized reduction from Clique parameterized by k to the Hiding Leader problem parameterized by $d + k$. The Clique problem parameterized by k is known to be $W[1]$-hard [4]. This means that it is unlikely that there exists an fixed-parameter tractable algorithm for the problem parameterized by the solution size k. The parameterized reduction from Clique parameterized by k to the Hiding Leader problem parameterized by $d + k$ implies that it is unlikely for the Hiding Leader problem parameterized by $d + k$ to have an fixed-parameter tractable algorithm.

CLIQUE
Given a graph G and an integer k, the goal is to determine whether the given graph G has a clique (refer Definition 1.2.9) of size at least k.

Now we prove the following hardness result.

Theorem 5.3.1. *The* HIDING LEADER *problem is W[1]-hard when parameterized by* $d + k$.

Proof. We present a parameterized reduction from the CLIQUE. Let (G, k) be an instance of CLIQUE problem where $G = (V, E)$. We construct an instance (H, L, C, k, d) where $H = (V', E')$ of HIDING LEADER problem the following way.

The set V' of vertices of H: For every $u_i \in V$, we create a single vertex u_i and $n - 1 - |N_G(u_i)|$ new vertices denoted by $X_i = \{x_{i,1}, \ldots, x_{i,n-1-|N_G(u_i)|}\}$. We add one additional vertex b. Let $|V| = n$. We then add $n + k$ vertices, named $L' = \{l_1, ..., l_{n+k}\}$. This completes the list of vertices in V'. So $V' = V \cup \bigcup_{i=1}^{n} X_i \cup \{b\} \cup L'$.

The set E' of edges of H: We add an edge between two vertices u_i and u_j of V in H if and only if $(u_i, u_j) \notin E(G)$. Make b adjacent to every vertex of V and make u_i adjacent to every vertex of X_i. Now we add edges (l_i, l_j) for all i, j except for the pair (l_1, l_2). Lastly we create two edges (l_1, b) and (l_2, b). This completes the list of edges in E'

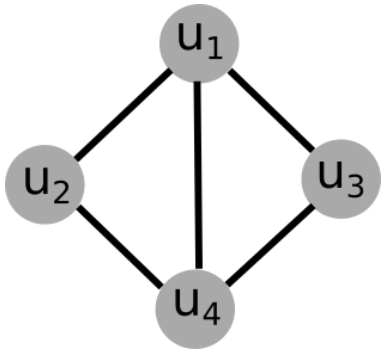

Figure 5.1: An instance of CLIQUE

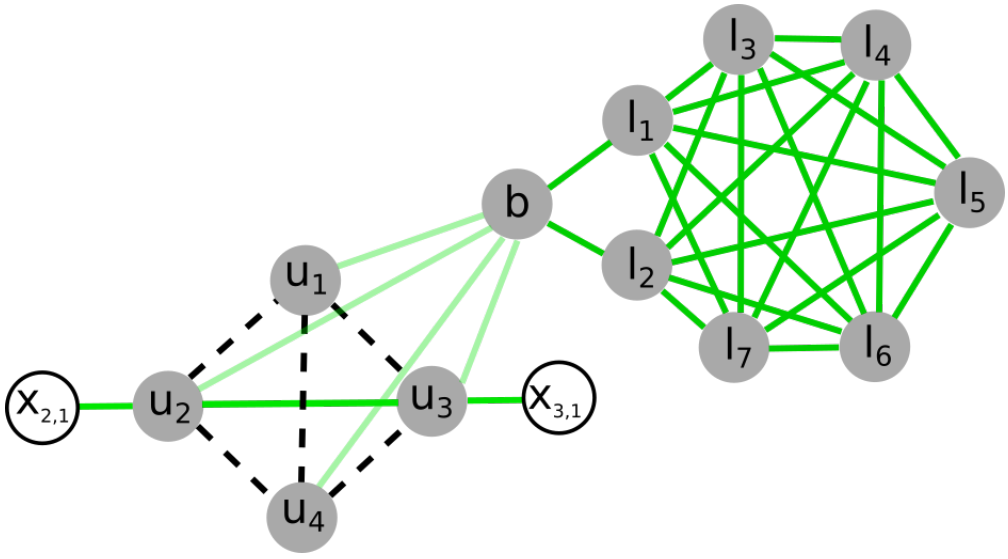

Figure 5.2: Reducing the instance of the CLIQUE problem in Figure 5.1 to an instance of the HIDING LEADER problem.

See Figure 5.4 for an illustration of the reduction. In Figure 5.4 green lines represent the edges of E' whereas dashed black lines represent the edges of E in G. We define

- $H = (V', E')$ where V' and E' are as per the construction.
- $L = V' \backslash V$
- $k' = \binom{k}{2}$, the budget of the problem
- $C(H, x) = d_H(x)$ for all $x \in V'$
- $d = k$ the safety margin.

Now we claim that G has a clique of size at least k if and only if (H, L, k', d) is a yes-instance. Assume first that (H, L, k', d) is a yes-instance. From the HIDING LEADER problem definition, we know that the edges to be added to H must be chosen from $F \times F$. Since in the constructed instance of the HIDING LEADER problem, we have $F = V' \backslash L$, using the definition of L we have $F = V' \backslash (V' \backslash V) = V$. Hence the edges must be added from $V \times V$, but given the definition of H, $(V \times V) \backslash E$ are already in H. Therefore the edges to be added to H must be taken from E. Let $E'' \subseteq E'$ be the solution set for the HIDING LEADER problem. Notice that for each $u \in F$, $d_H(u) = n$ by constructions. Notice how the highest

degree members of L are, in fact, the members of L' with their degrees equal to $n+k-1$. Therefore for E'' to be a solution, it must increase the degree of some k vertices of $F(=V)$ to at least $n+k-1$. Since all the vertices in V are of degree n, E'' must increase the degree of some k vertices in V by at least $k-1$. Since the budget $k' = \binom{k}{2}$ we can only increase the degrees of k vertices in V by exactly $k-1$. Therefore the total degree increase in $k(k-1) = 2\binom{k}{2}$. This implies that each edge added must increase the degree of two vertices. The only possible solution is if the set E'' forms a clique in G since all edges in E'' are taken from G. Since the HIDING LEADER instance is a yes instance, such a clique exists in G. Therefore HIDING LEADER is a yes-instance implies there exists a $k-$clique in G.

Now assume that G contains a k-clique. Consider the set of vertices D of such a k-clique. Notice that set of all the possible edges between these vertices, say E'', is a subset of E. Hence all the edges in E'' are not in H and can be added in the solution. Consider the case where we add all the edges from E'' to H. $|S| = \binom{k}{2}$. So adding all edges of E'' is within the budget of HIDING LEADER instance. All vertices of D are of degree n in H. Since each vertex of D is adjacent to $k-1$ edges in E'', adding all edges of E'' to H will increase the degree of every vertex of D to $n+k-1$. We know that the degree of the highest-degree leader is $n+k-1$ in HIDING LEADER instance. So adding these edges from E'' to H increases the degree of $|D| = k$ vertices to that of the highest-degree leader while staying within budget. Hence the HIDING LEADER instance is a yes instance. So G contains a k-clique implies (H, L, C, k, d) is a yes instance. Therefore G contains a clique of size at least

This concludes the proof of our claim. Therefore there exists a parameterized reduction from the CLIQUE problem to the HIDING LEADER problem. In this reduction

1. (G, k) is a yes instance of the CLIQUE problem if and only if the constructed instance (H, L, C, k, d) is a yes instance of the HIDING LEADER problem.

2. $d + k' = k + \binom{k}{2} \leq 2k^2$

3. The reduction runs in $|x|^{O(1)}$. It is a polynomial-time reduction.

Therefore the reduction is a parameterized reduction (refer to Definition 4.1.1). Hence the HIDING LEADER problem parameterized by $d+k$ is unlikely to have a fixed-parameter tractable algorithm (refer to Theorem 4.1.1) as CLIQUE problem is assumed to be not in

FPT. Since the parameterization $d+k$ is at least $W[1]$-hard for the HIDING LEADER problem, it implies that parameterizations by d or k individually is at least $W[1]$-hard as well. □

5.4 The HIDING LEADER problem parameterized by Δ+d

We consider the HIDING LEADER problem for the bounded degree graphs (maximum degree is Δ) and the degree centrality. The problem in the setting of bounded degree graphs becomes FPT when parameterized by $\Delta + d$. Since the HIDING LEADER problem is at least as hard as the CLIQUE problem, this gives a basis to believe that adding Δ as one of the parameters to the HIDING LEADER problem might make it fixed-parameter tractable. Hence we consider the problem parameterized by $\Delta + d$. As we will see, this does make the problem fixed-parameter tractable by fetching us a kernel. Let us see a step-wise approach to motivate and reach the kernel.

5.4.1 Naive solution to the HIDING LEADER problem

The aim is to solve the HIDING LEADER problem using as few edges as possible. Let S be the set of d vertices whose degree is to be made greater than or equal to the degree of any of the leaders by adding at most k edges as the solution set. Notice that given a solution set $S = \{u_1, u_2, ...u_d\}$, we add edges to the vertices of S. It would be ideal if most of them are internal edges, that is, both the endpoints of the newly added edge (u, v) are in S. This is because an internal edge increases the degree of two vertices of S, whereas a non-internal edge only increases the degree of one vertex of S. Thus more internal edges will mean better use of the k edges available to us to add.

Therefore, a solution to the HIDING LEADER problem is a set of d vertices that maximizes

a) the ability to add internal edges

b) the degree sum of the set of vertices

An independent set (IS) maximizes the ability to add internal edges, and a weighted independent set (WIS) with weights of vertices given by the degree of the vertices maximizes both (a) and then maximizes (b). So such a weighted independent set can be thought of as a naive solution to the problem.

This solution is not necessarily the optimal solution, as adding internal edges to the solution set beyond a point is just as good as adding external edges. For example, consider a solution set S where we have added k' edges already, and few of the vertices of S have a degree greater than or equal to that of the highest degree leader (say of degree Δ). So adding an internal edge to a vertex with degree Δ is as good as adding an external edge to S.

5.4.2 Independent Set parameterized by $\Delta + d$

We now compute a solution to the Independent Set problem parameterized by $\Delta + d$.

Independent Set
Given a graph G and a positive integer d, the Independent Set problem is to determine whether the given graph G has an independent set of size at least d (refer Definition 1.2.12).

Now we prove the following result.

Lemma 5.4.1. *The* Independent Set *problem parameterized by* $\Delta + d$ *admits a kernel of size* $(d-1)(\Delta+1)+1$.

Proof. Let G be a graph with maximum degree Δ. Let u be an arbitrary vertex in G. Let S be a set of $\Delta + 2$ vertices in G such that $u \in S$. Then there exists a vertex $v \in S$ that is non-adjacent to u. Note that u can have at most Δ neighbours in S. As S is of size $\Delta + 2$, there is a vertex v in S that is non-adjacent to u.

Now consider any set of $2(\Delta+1)+1$ vertices. For any vertices from this set, there always exists a non-adjacent vertex. For any two vertices of this set, there exists a vertex non-adjacent to both of them. The reason is this. Suppose u and v are two non-adjacent vertices in the set. Since the maximum degree is Δ, we have $|N[u]| + |N[v]|$ at most $2(\Delta+1)$. Since

the set is of size $2(\Delta+1)+1$, there always exists a vertex w in the set which is non-adjacent to both u and v.

Extending this logic for any set S of $(d-1)(\Delta+1)+1$ vertices, we have that given any $d-1$ pair-wise non-adjacent vertices, we have one vertex independent of all of them. So if we have an independent set of size $d-1$ in the set, then we have an independent set of size d in this set of vertices.

Let S be a set of $(d-1)(\Delta+1)+1$ vertices in G. If S has an independent set of size $d-2$ in S, then S has an independent set of size $d-1$ and subsequently an independent set of size d. So if we have an independent set of size 2, then we have an independent set of size d, but we already showed that we always have an independent set of size 2 in this set of vertices. Therefore in any set of $(d-1)(\Delta+1)+1$ vertices we have an independent set of size d.

This means that given a graph G, we only need to look at any $(d-1)(\Delta+1)+1$ vertices to find an independent set. This essentially reduces the number of vertices we need to search for an independent set to $(d-1)(\Delta+1)+1$ vertices, and this reduction in number can be achieved in polynomial time. *Therefore* Independent Set *parameterized by* $\Delta+d$ *admits a kernel of size* $(d-1)(\Delta+1)+1$. □

This completes the kernelization of the Independent Set problem parameterized by $\Delta+d$. We can now brute force the kernel and achieve an independent set of size d in FPT time when parameterized by $\Delta+d$, more precisely in $\binom{(d-1)(\Delta+1)+1}{d}$ time. However, a better algorithm exists when we have the existence of a kernel of size $(d-1)(\Delta+1)+1$.

Algorithm for Independent Set

1. Pick a random vertex in the kernel and add it to the solution set I.
2. Remove the chosen vertex and its neighbors from the kernel.
3. If $|I| < d$, repeat steps 1 and 2.

In this algorithm, at each iteration Step 1 takes $O(1)$ time. In total, there are d iterations of the first step, so in total, Step 1 takes $O(d)$ time. Each iteration of Step 2 takes $O(\Delta+1)$

time. So in total, Step 2 takes $d(\Delta + 1)$. So the algorithm runs in $O(d\Delta)$ time.

This is a polynomial-time algorithm. Such an algorithm exists due to the following strong assumptions:

1. $d_G(v) \leq \Delta$ for all v
2. the existence of $(d-1)(\Delta+1)+1$ kernel

In cases where either of the assumptions is not true, the existence of a polynomial time algorithm would mean $P = NP$.

5.4.3 Lower Bound for Maximum Independent Set

In a bounded degree graph, we can color the vertices of the graph using at most $\Delta + 1$ color such that no two adjacent vertices are of the same color. This gives us a lower bound on the size of the maximum independent set (MIS) of the graph. We can observe that vertices of the same color form an independent set. So by the pigeonhole principle, we have at least one color which is used for at least $\frac{|V(G)|}{\Delta+1}$ vertices. That is, the maximum independent set of the graph has size at least $\frac{|V(G)|}{\Delta+1}$.

The size of the kernel also gives us a lower bound on the size of the independent set of a graph with a bounded degree. Let $|V(G)| = n$. Since for the guarantee of an independent set of size d, we need a kernel to exist, we have that

$$\begin{aligned}(d-1)(\Delta+1)+1 &\leq n \\ d-1 &\leq \frac{n-1}{(\Delta+1)} \\ d &\leq \frac{n+\Delta}{(\Delta+1)} \\ d_{max} &= \left\lfloor \frac{n+\Delta}{(\Delta+1)} \right\rfloor\end{aligned}$$

Here d_{max} is the maximum possible d satisfying the inequality. Note that d_{max} is the size of an independent set. This is a tighter lower bound than the one obtained earlier using

the coloring argument. We can construct a graph example where d_{max} is the size of the maximum independent set but $\frac{n}{\Delta+1}$ is 1, lower than the size of a maximum independent set. Consider a graph with l disjoint components; $l-1$ of the components are $\Delta+1$ size cliques, and the last component is an isolated vertex. Here $|V(G)| = n = (l-1)(\Delta+1)+1$. We observe that an independent set of size d_{max} exists, where

$$d_{max} = \left\lfloor \frac{n+\Delta}{(\Delta+1)} \right\rfloor = \left\lfloor \frac{(l-1)(\Delta+1)+\Delta+1}{(\Delta+1)} \right\rfloor = l.$$

The bound obtained using the coloring argument is

$$\left\lfloor l - \frac{\Delta}{\Delta+1} \right\rfloor \;=\; l-1.$$

Since each of the components in the constructed graph is a clique or an isolated vertex, we can have at most one vertex from each component. Therefore $|MIS| \leq l$ and $|MIS| \geq l$, this implies $|MIS| = l$

5.4.4 Weighted Independent Set parameterized by $\Delta + d$

In the process of coming up with a naive solution for Hiding Leader problem, we try to obtain a weighted independent set of maximum weight where the weights are given by the degree of the vertices. Following the result obtained in the case of the Independent Set problem parameterized by $\Delta + d$, we will try to obtain something similar in the case of the Weighted Independent Set problem parameterized by $\Delta + d$.

Weighted Independent Set
Given a graph G, a weight function $w : V \to \mathbb{R}$ and a positive integer d, the problem is to find an independent set I of size d in G that maximizes $\sum_{u \in I} w(u)$.

Consider a vertex-weighted graph G where the weights are given by $w(x)$, $x \in V(G)$. We know that in any set of $(d-1)(\Delta+1)+1$ vertices we have an independent set of size at least d. To maximize the weights, a naive approach is to maximize the weight of these $(d-1)(\Delta+1)+1$ vertices. Let us order the vertices according to their weights and pick the $(d-1)(\Delta+1)+1$ vertices of higher weights. Let us call this set S.

Lemma 5.4.2. *A maximum weighted independent set of size d always contained in S.*

Proof. For the sake of contradiction, suppose that some vertices of a maximum weighted independent set I' of size d obtained by brute force algorithm lie outside of S. Notice that any vertex $u \in S$ satisfies $w(u) \geq w(v)$ where $v \notin S$. Consider the sets $Q = I' \cap S$ and $P = I' \setminus Q$. Using Q as a $|Q|$ sized weighted independent set, we construct a weighted independent set of size d contained in S and call this Q' an example of such a construction can be seen in figure 5.3. Note that we have shown previously that this is always possible. For all vertices $u \in S$ and $v \in P$, we have $w(u) \geq w(v)$. Thus

$$\sum_{u \in Q' \setminus Q} w(u) \geq \sum_{v \in P} w(v).$$

Note that $Q' = (Q' \setminus Q) \cup Q$ and $I' = P \cup Q$. This implies

$$\sum_{u \in Q'} w(u) \geq \sum_{v \in I'} w(v).$$

Therefore Q' is a weighted independent set contained in S and it has weight larger than or equal to the weight of I', a contradiction to the assumption that I' is a maximum weight independent set of size d. □

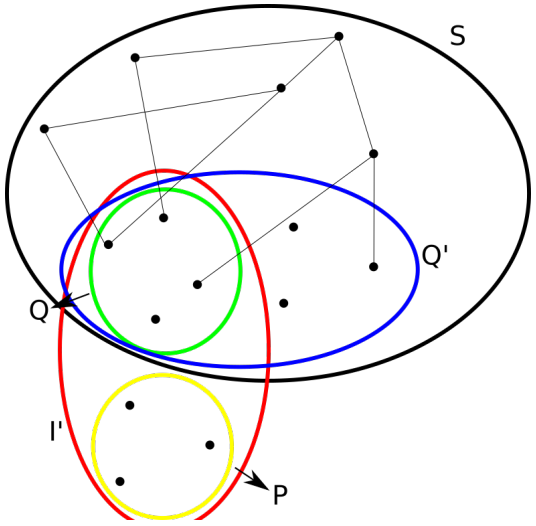

Figure 5.3: Example of the construction of Q' from given I' and S

This shows that it is sufficient to consider the top $(d-1)(\Delta+1)+1$ weighted vertices to find a maximum weighted independent set of size d in G. The top $(d-1)(\Delta+1)+1$ weighted vertices can be fetched in polynomial time. This is enough to show the Weighted Independent Set problem parameterized by $\Delta+d$ admits a kernel of size $(d-1)(\Delta+1)+1$.

This completes the kernelization of the Weighted Independent Set problem when parameterized by $\Delta+d$. We can now brute force the kernel and achieve a maximum weighted independent set in FPT time when parameterized by $\Delta+d$, more precisely in $\binom{(d-1)(\Delta+1)+1}{d}.d^{O(1)}$ time.

However a better algorithm exists to achieve the same result. Pick a vertex u with the maximum weight in the kernel. Given a weighted independent set of size d, we can replace the lowest weight vertex in the weighted independent set with u to obtain an equal weight or higher weight independent set. This is not possible if and only if the weighted independent set contains a neighbor of u. This implies that every weighted independent set of size d either contains u or some vertices from $N_G(u)$. This gives us a branching algorithm.

Algorithm to find Weighted Independent Set

1. Pick a vertex u with the highest weight in the kernel. In one branch add u to the solution set, and in the other branches add one of its neighbors to the solution set.
2. In each of the branches, remove the chosen vertex and its neighbors from the kernel.
3. Repeat steps 1 and 2 in each of the cases.
4. Repeat step 3 until the size of solution set in each branch is d
5. Compare the weights of all the solution sets to obtain a maximum weighted independent set of size d.

In the first step of the algorithm, there are at most $\Delta+1$ branches. In subsequent iterations of the first step, each of the $\Delta+1$ steps in the previous iteration of the first step has $\Delta+1$ branches each. So by the second iteration of step 1, we have $(\Delta+1)^2$ total branches. So by k iterations of step 1 (or $k-1$ iterations of step 3) we have $(\Delta+1)^k$ branches. In each iteration of step 1, the solution size increases by one. So for a d size solution we have $(\Delta+1)^d$ many branches.

Each iteration of second step takes at most $(\Delta+1)^{O(1)}$ time. In total Step 2 is iterated over $(\Delta+1)^d$ times. To compute a maximum weighted independent set of size d, we have to compare $(\Delta+1)^d$ solution sets and pick the top-weighted set. Therefore the algorithm runs in $O((\Delta+1)^d \cdot (\Delta+1)^{O(1)})$ time. This is a better running time than $\binom{(d-1)(\Delta+1)+1}{d} \cdot d^{O(1)}$.

This algorithm runs in the same time for an unkernelized graph G and is provided for the sake of completeness.

5.4.5 Kernelization of the Hiding Leader problem

Consider a vertex-weighted graph G with weights given by $w(x) = d_G(x)$. By Lemma 5.4.2, we know a maximum weighted independent set of size d contains vertices from $(d-1)(\Delta+1)+1$ highest weight vertices of F. This is the naive solution. If there exists a solution, we would only consider it if it's better than the naive solution. This is the motivation to believe that a solution better than the naive solution might also be contained within $(d-1)(\Delta+1)+1$ followers of the highest weights. In other words, the solution to the Hiding Leader problem might also be contained in the top $(d-1)(\Delta+1)+1$ followers. We will show this exact result in the next theorem.

Let us order the vertices of F according to decreasing order of their degrees and pick the first $(d-1)(\Delta+1)+1$ vertices, the vertices of high degrees. Let's call this set S.

Theorem 5.4.3. *The solution to the* Hiding Leader *problem is contained in S. That is, the* Hiding Leader *problem admits a kernel of size $(d-1)(\Delta+1)+1$.*

Proof. For the sake of contradiction assume that some vertices of the solution to the Hiding Leader problem obtained by brute force algorithm lie outside of S. Let the solution set obtained by brute force be F'. We assume that there exists $u \in F'$ such that $u \notin S$.

Notice that any vertex $u \in S$ satisfies $w(u) \geq w(v)$ where $v \notin S$. Consider the sets $Q = F' \cap S$ and $P = F' \setminus Q$. Thus $F' = Q \cup P$. Using Q as a $|Q|$ sized set, construct a larger set by adding $d - |Q|$ pair-wise non-adjacent vertices from S which are nonadjacent to every vertex of Q. We call this set of vertices added to Q Q^+; it is of size $d - |Q|$ and it lies in S. We have previously shown that this is always possible. For all vertices $u \in S, v \in P$, we have

$w(u) \geq w(v)$. So

$$\sum_{u \in Q^+} w(u) \geq \sum_{v \in P} w(v).$$

Therefore we have

$$\sum_{u \in Q \cup Q^+} w(u) \geq \sum_{v \in Q \cup P = F'} w(v).$$

This result is similar to the one obtained in Lemma 5.4.2. We will now use a bijection from P to Q^+ to show that if F' is a solution to the HIDING LEADER problem then $Q \cup Q^+$ is also a solution to the HIDING LEADER problem.

Let $\phi : P \to Q^+$ be any bijection. For any assignment of edges E' that is obtained by brute force technique, we can construct an assignment of edges E'' for $Q \cup Q^+$. Let $(u, v) \in E'$, then

- if $u, v \notin P$ include (u, v) in E''.
- without loss of generality if only $u \in P$ include $(\phi(u), v)$ in E''.
- if $u, v \in P$ include $(\phi(u), \phi(v))$ in E''.

Here E'' is an assignment of edges that solves $Q^+ \cup Q$. Figure 5.4 shows such a construction. So if E' makes the degrees of all vertices of Q in brute force solution greater than or equal to the degree of any of the leaders, then so does E'' in $Q \cup Q^+$. Notice that for all vertices $u \in Q^+, v \in P$, we have $w(u) \geq w(v)$. As $\phi(u) \in Q^+$, we have that $w(\phi(u)) \geq w(u)$ for all $u \in P$. Also, notice that for any edge $(u, v) \in E'$, there exists $(\phi(u), v)$ or $(\phi(u), \phi(v))$ in E'', that is the number of edges incident to a vertex $u \in P$ in E' is equal to the number of edges incident to $\phi(u) \in Q^+$ in E''. Therefore if E' increases the degree of a vertex $u \in P$ to make its degree greater than or equal to the degree of any of the leaders, then so does E'' for $\phi(u)$.

Since E' is an assignment of edges obtained by brute force technique that solves the HIDING LEADER problem, E'' is an assignment of edges that solves $Q \cup Q^+$. Hence $Q \cup Q^+$ is also an equivalent solution. Therefore a solution to the HIDING LEADER problem always lies inside S.

Since we are guaranteed that a solution set to any instance of the problem always lies inside

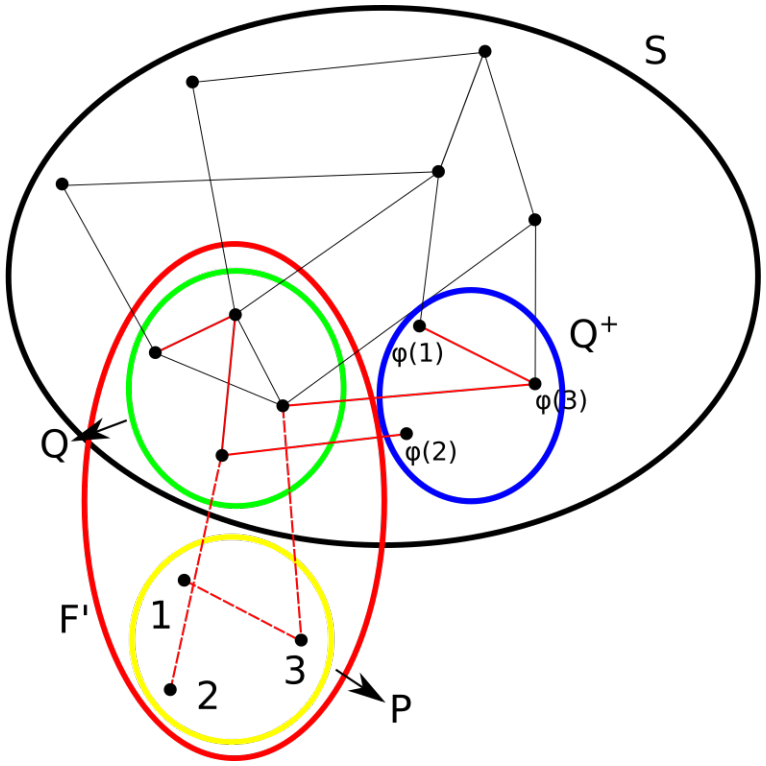

Figure 5.4: Example of the construction of Q^+ from given F' and S. The solid red lines are part of constructed edge set E'' and dashed lines are part of E'

S, we only need to check vertices of S to obtain a solution. Given an instance of Hiding Leader, we can fetch an appropriate set S in polynomial time. This is sufficient to show that the Hiding Leader problem parameterized by $\Delta + d$ admits a kernel of size $|S| = (d-1)(\Delta+1)+1$. □

Since the kernel size is $O(d\Delta)$, a brute force algorithm runs in FPT time. For each d vertex subset of S, we can compute all possible internal edge configurations (all possible adjacency of the vertices of d-subset of S) in 2^{d^2} ways. For each of the internal edge configurations, for a vertex u in the d-subset that has its degree less than that of the highest degree of the leaders'

vertices, we can add edges from outside until their degree is equal to the highest degree of the leaders. This takes $d.n^{O(1)}$ time. In total, there are $\binom{|S|}{d} \leq (2d\Delta)^d$ many d-subsets of S. So the total running time of the brute force algorithm is $2^{d^2+d}(d\Delta)^d d \cdot n^{O(1)} = 2^{d(d+1)}(d\Delta)^d n^{O(1)}$ which is FPT-time. Therefore the hiding leader problem parameterized by $\Delta + d$ is fixed-parameter tractable.

5.4.6 Assignment of edges solving a d vertex set

We know we can brute force the kernel let us see if we can improve the running time to find a satisfying assignment of edges E' that solves a d vertex set D, using the least number of edges. That is, our goal is to find E' so that the degrees of all the vertices in D are greater than or equal to any of the leaders in $G' = (V, E \cup E')$.

Let the highest degree of the leaders be d_L. Consider the case where D is an independent set. Let $\{v_1, v_2, ..., v_d\}$ be the vertices of D. We need degrees of $v_i \in D$ to be greater than or equal to d_L. We say that a vertex v_i has a residual degree rd_i if it requires rd_i edges to be added to it to make its degree d_L. That is $rd_i = d_L - d_G(v_i)$. Re-index v_i's such that sequence $\{rd_i\}$ is a non-increasing sequence. We safely assume that all rd_is are positive integers as if a vertex with a degree greater than or equal to d_L exists in D, we can simply remove it from D and reduce d to $d-1$ in our instance of the Hiding Leader problem.

Note that if it is possible to find the required edge set E' such that all the edges are from $D \times D$, then we would have constructed a graph with degree sequence $\{rd_1, rd_2, ..., rd_d\}$ using the vertices of D. Taking inspiration from this insight, we try to construct a graph with degree sequence $\{rd_1, rd_2, ..., rd_d\}$ for a given independent set D. To this end, we use the Havel-Hakimi algorithm [13, 15] that, given a realizable degree sequence, constructs a graph with that degree sequence. A realizable degree sequence is a degree sequence for which at least one simple graph exists with that degree sequence. Not all degree sequences are realizable. An example of such a sequence is $\{4, 4, 2, 2, 1\}$. A simple example of a realizable degree sequence is $\{2, 2, 2\}$, a 3 clique.

In the Havel-Hakimi algorithm, given a realizable degree sequence $s, x_1, x_2, ..., x_n$ it outputs a smaller realizable degree sequence $x_1 - 1, x_2 - 1, ..., x_s + 1, x_{s+1}, ..., x_n$ and takes that sequence as an input to the algorithm. Essentially it takes the vertex with the most residual

degree s and adds s edges from that vertex to the next s vertices when ranked according to their residual degrees. There is a generalized Havel-Hakimi algorithm [20] where instead of picking the top residual degree vertex, we pick any vertex with residual degree r and add r edges between it and the top r residual degree vertices other than itself. The Havel-Hakimi algorithm and the generalized Havel-Hakimi algorithm work for all realizable degree sequences.

We borrow the steps from the generalized version of the Havel-Hakimi algorithm and use them for trying to construct a graph given a residual degree sequence $\{rd_1, rd_2, ..., rd_d\}$. We pick a v_i vertex with residual degree rd_i and add edges between it and the top rd_i residual degree vertices and remove v_i from D and re-index set of vertices of D so that their residual degrees from a non-increasing sequence. We remove any vertex whose residual degree becomes 0. We repeat the aforementioned steps. If after repeating these steps, at most one vertex remains in D we stop. Now for an independent set D, if we can create such a graph, we have found an appropriate edge set E'. The Havel-Hakimi algorithm can be optimized to run in $O(n^2 \log n)$ time which is $O(d^2 \log d)$ in our case.

However, the sequence $\{rd_i\}$ need not be a realizable degree sequence. In such cases, we create a new degree sequence by adding 1s to the end of degree sequence $\{rd_i\}$. Notice that when $\sum_i rd_i$ 1s are added to the degree sequence $\{rd_i\}$ it is always realizable. Each 1 represents an edge from vertices of D to vertices outside D. We need to minimize the edges added hence we need to find the minimum number of 1s which, when added to $\{rd_i\}$, makes it a realizable degree sequence. To generate a graph, we start by running the Havel-Hakimi algorithm for $\{rd_i\}$. If the algorithm fails, we add a 1 to the end of $\{rd_i\}$ and try to generate a graph again. If the algorithm fails again, we add another 1 to the degree sequence and keep doing so until we can create a graph. After such a graph is created, we find appropriate random vertices from $V \setminus D$ to which we can transfer the edges between vertices of D and the 1s added to $\{rd_i\}$. This takes $O((dd_L)^3 \log(dd_L))$ time.

Let us try to extend this idea to any set D. If we follow the same approach as before, we will fail because some vertices of D are already connected to some other vertices of D, meaning in our residual degree sequence $\{rd_i\}$ generated by vertices of D, there are some restrictions for each vertex of D in terms of the vertices it can be connected to achieve a graph. We use the theorem proved in [19] by Hyuunju Kim et al. to develop an algorithm to generate a graph given restrictions on the vertices in terms of the vertices it can be connected

to.

Let vertex $v \in V$ with restriction set $X(v) = \{i_1, i_2, ..., i_m\}$ where $i_j \in V$ and $i_j \neq v \ \forall j$ where $G(V, E)$ is such the $E = \phi$ and vertices of V are associated with residual degrees $\{d_1, d_2, ...d_n\}$ arranged in non decreasing order. Let the residual degree of v be d_v. The theorem by Hyuunju Kim et al. in [19] shows that a simple graph realizing the degree sequence $\{d_1, d_2, ..., d_n\}$ given restriction set $X(v)$ for a vertex v exists if and only if the degree sequence produced by reducing the residual degrees of top d_v residual degree vertices not in $X(v) \cup \{v\}$ by 1 and removing d_v is realizable.

This gives us an algorithm to generate graphs for the residual degree sequence of any set D where each vertex, u of D, has a restriction set $X(u) = N_{G[D]}(u)$. For all choices of the first vertex j with residual degree rd_j, we produce a new residual degree sequence by reducing the residual degrees of top rd_j residual degree vertices not in $X(i) \cup \{i\}$ by 1 and removing j from D. We remove any vertices from D whose residual degrees are 0. For the new residual degree sequences obtained from the different Ds produced, we repeat the previous steps until we cannot repeat them anymore. That is, the remaining vertices form a clique (a single vertex is a 1 clique). We add these vertices to the graphs produced in the previous steps. We have $d!$ many graphs produced. Pick the graph where the sum of the residual degrees of its vertices is minimum. Now for all the vertices in this graph, add edges between the vertices with a non-zero residual degree and random vertices in G of our Hiding Leader problem until all the residual degrees become 0. We have obtained the required edge set E'. When optimised this takes $O(d! \log d)$ time.

5.5 Centrality given by core centrality

We will consider instances of the Hiding Leader problem where the centrality measure of a vertex is defined by the core centrality. Unlike the degree centrality, which just gives information about the number of edges of a vertex, the core centrality of a vertex contains information about a vertex and some of its neighbors. A vertex is a component of some dense, close-knit structure if its core centrality is high within the graph.

Definition 5.5.1. *Let $r \geq 2$ be a fixed integer. The r-core of a graph G is the largest induced*

subgraph of G in which all vertices have a degree of at least r.
$C_{core}(v) = max\{r : v \text{ belongs to some } r\text{-core}\}$

From Definition 5.5.1, we can infer that given a vertex u with core centrality r it has at least r vertices in $N_G(u)$ whose core centrality and degree centrality are greater than or equal to r. If we consider the vertices of a graph as people belonging to a group and edge (u, v) as person u having an influence on v and vice versa. Then in such a graph, a high-degree vertex is one that has an influence on a lot of people. We call them high-influence people. A vertex with high core-centrality, however, is one that has an influence on a lot of high-influence people.

The hiding leader problem when core centrality is considered is proven to be NP-hard using a reduction from the Set Cover problem even when the highest centrality of the leaders is 3 [5]. It is also proven that the Set Cover problem parameterized by solution size is $W[2]$ using a parameterized reduction from the Dominating Set problem parameterized by solution size[4]. So it would seem as though we have a parameterized reduction from Dominating Set parameterized by solution size to the Hiding Leader problem parameterized by k, d or $k + d$. Use the parameterized reduction to Set Cover problem from Dominating Set and then the polynomial reduction from the Set Cover problem to Hiding Leader problem. However, that is not a direct conclusion we can arrive at. This is because the polynomial reduction used in proving that NP-hardness of Hiding Leader problem is not a parameter preserving reduction. That is, k, d or $k + d$ are not bounded by any computable function g(solution size of set cover problem) in the polynomial reduction.

The hiding leader problem in the core-centrality setting is similar to the Edge k-Core (EkC) problem defined in [2].

Edge k-Core
Given a graph G and three positive integers d, k and b the Edge k-Core problem is to produce a k-core of size at least d by adding at most b edges to the graph.

Consider an instance of the Hiding Leader problem with core-centrality. If the set of leaders is disconnected from the set of followers, that is, no edges $(u, v) \in E$ such that $u \in L$ and $v \in F$, then this instance is essentially an Edge k-Core instance with k equals highest centrality of leaders. To avoid confusion, we use parameter k' for the budget of the hiding

leader problem, that is, the number of edges that can be added.

A parameterized reduction from the CLIQUE problem to the EDGE k-CORE problem parameterized by $d + b$ is given in [2]. This would imply that the HIDING LEADER problem in the setting of core centrality is $W[1]$-hard when parameterized by $k' + d$.

Chapter 6

Conclusion

The theory of parametrized complexity is a very useful toolbox to look at different naturally arising parameters and their influence on problems. More importantly, if a particular parameterization of a NP-hard problem is known to be fixed-parameter tractable, then for a small parameter size, we achieve a dramatic improvement in the running time of an algorithm solving the problem (orders of magnitude apart). The scope of discussion is not just limited to the running time of an algorithm solving the problem but also the pre-processing procedures one can apply to the parameterized versions of the problem. The toolbox comes with its own unique algorithm techniques, like bounded search trees. Since it is a relatively new field in computer science not much is proved in the intractability of problems but making a few motivated, strong assumptions give rise to various results in intractability not based on P$\neq$NP. The field is full of interesting insightful problems when looked under the lens of parameterized complexity, an example of which is the Hiding Leader problem belonging to the class of edge manipulation problems in graphs. The results of the Hiding Leader problem can have interesting consequences on how one studies covert networks and can be insightful for the field of Social Network Analysis.

In this thesis, we got the motivation to look at parameterized complexity of some hard problems using real-world examples and looked at a few kernels and kernelization techniques. We looked at the bounded search tree method for solving parameterized problems and applied it to the specific case of the vertex cover problem. We got an insight into solving recursions using the branching vectors. We defined the polynomial-time reductions equivalent machinery for studying fixed-parameter intractability and came across a few examples that are believed to be in $W[1]$. We went through an overview of the W-hierarchy and some of its results. We defined the NP-complete problem of the Hiding Leader problem and studied it at length for the setting of degree centrality, coming up with original results and interpretation of the previously shown result. We briefly looked at the setting of core-centrality.

Future Direction

Although some new results have been obtained in the Hiding Leader problem, a lot of interesting problems remain unsolved. In the setting of degree, centrality graphs of bounded treewidth are a compelling subsetting to explore as they yield a fixed-parameter tractable algorithm for the weighted independent set problem, which, as we discussed, can be considered a precursor to the Hiding Leader problem. Graphs that admit clique separators are another class of graphs for which the problem might prove to be tractable. In the setting of core-centrality, no tractability or intractability results are fully known. Since the degree centrality setting becomes tractable in bounded-degree graphs, the core-centrality setting might also lead to similar results in that setting. Graphs that admit clique-separators might prove to be interesting for core-central due to the fact that cliques are the most basic examples of cores of a graph. One can also look at other centrality settings like betweenness centrality and closeness-centrality, both of which are NP-hard problems.

Bibliography

[1] Abrahamson, K.R., Downey, R.G., Fellows, M.R.: Fixed-parameter tractability and completeness IV: On completeness for W[P] and PSPACE analogues. Ann. Pure Appl. Logic 73(3), 235–276 (1995)

[2] Chitnis, R., Talmon, N. (2018). Can We Create Large k-Cores by Adding Few Edges?. In: Fomin, F., Podolskii, V. (eds) Computer Science – Theory and Applications. CSR 2018. Lecture Notes in Computer Science(), vol 10846. Springer, Cham. https://doi.org/10.1007/978-3-319-90530-3_8

[3] Cormen, T.H., Leiserson, C.E., Rivest, R.L. and Stein, C.. Introduction to Algorithms, MIT Press, 2022.

[4] Cygan, M. & Fomin, F. & Kowalik, L. & Lokshtanov, D. & Marx, D. & Pilipczuk, M. & Pilipczuk, M. & Saurabh, S. (2015). Parameterized Algorithms. 10.1007/978-3-319-21275-3.

[5] Dey, P. and Medya, S.. 2019. Covert Networks: How Hard is It to Hide? In Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS '19). International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 628–637.

[6] Dey, P. and Medya, S.. 2020. Manipulating Node Similarity Measures in Networks. In Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS '20). International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 321–329.

[7] Downey, R.G., Fellows, M.R.: Fixed-parameter intractability. In: Proceedings of the 7th Annual Structure in Complexity Theory Conference, pp. 36–49. IEEE Computer

[8] Downey, R.G., Fellows, M.R.: Fixed-parameter tractability and completeness. Proceedings of the 21st Manitoba Conference on Numerical Mathematics and Computing. Congr. Numer. 87, 161–178 (1992)

[9] Downey, R.G., Fellows, M.R.: Fixed-parameter tractability and completeness I: Basic results. SIAM J. Computing 24(4), 873–921 (1995)

[10] Downey, R.G., Fellows, M.R.: Fixed-parameter tractability and completeness II: On completeness for W[1]. Theoretical Computer Science 141(1 & 2), 109–131 (1995)

[11] Downey, R. G. and Fellows, M. R. 2013. Fundamentals of Parameterized Complexity. Springer Publishing Company, Incorporated.

[12] Garey, M., Johnson, D., Computers and Intractability. A Guide to the Theory of NP-Completeness (Freeman, San Francisco, 1979)

[13] Hakimi,S.L.: On the realizability of a set of integers as degrees of the vertices of a simple graph. J. SIAM Appl. Math. 10 (1962), 496–506.

[14] Hall, P.: On representatives of subsets. J. London Math. Soc. 10, 26–30 (1935)

[15] Havel, V.: A remark on the existence of finite graphs. (Czech), ˇCasopis Pˇest. Mat. 80 (1955), 477–480.

[16] Helfstein, S., & Wright, D. (2011). Covert or Convenient? Evolution of Terror Attack Networks. The Journal of Conflict Resolution, 55(5), 785–813. http://www.jstor.org/stable/23049885

[17] Hopcroft, J.E., Karp, R.M.: An n 5/2 algorithm for maximum matchings in bipartite graphs. SIAM J. Computing 2, 225–231 (1973)

[18] Kőnig, D.: Über Graphen und ihre Anwendung auf Determinantentheorie und Mengenlehre. Math. Ann. 77(4), 453–465 (1916)

[19] Kim, H., Toroczkai, Z., Miklós, I., Erdős, P. L., Székely, L. A.: On realizing all simple graphs with a given degree sequence.

[20] Mihail, M., Vishnoi, N. : On Generating Graphs with Prescribed Degree Sequences for Complex Network Modeling Applications, Position Paper, ARACNE (Approx. and Randomized Algorithms for Communication Networks), Rome (2002).

[21] Regan, K., Finite substructure languages, in Proceedings of Fourth Annual Structure in Complexity Conference, University of Oregon, Eugene, Oregon, June 19–22, 1989 (IEEE Comput. Soc., Los Alamitos, 1989), pp. 87–96

[22] Waniek, M., Michalak, T. P., Rahwan, T., and Wooldridge, M.. 2017. On the Construction of Covert Networks. In Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems (AAMAS '17). International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 1341–1349.

[23] West, D. B., Introduction to Graph Theory, Prentice Hall, 2000.