EXACT ALGORITHMS FOR OPTIMIZATION AND PARAMETERIZED VERSIONS OF SOME GRAPH THEORETIC PROBLEMS

By

Saket Saurabh

The Institute of Mathematical Sciences, Chennai.

A thesis submitted to the
Board of Studies in Mathematical Sciences

In partial fulfillment of the requirements

For the Degree of

DOCTOR OF PHILOSOPHY

of

HOMI BHABHA NATIONAL INSTITUTE

December, 2008

Homi Bhabha National Institute

Recommendations of the Viva Voce Board

As members of the Viva Voce Board, we recommend that the dissertation prepared by **Saket Saurabh** entitled "Exact Algorithms for Optimization and Parameterized Versions of Some Graph Theoretic Problems" may be accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy.

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

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

-- **Date :**
Guide : V. Raman

DECLARATION

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

Saket Saurabh

. . . to my mother.

ACKNOWLEDGEMENTS

First and foremost I would like to express my sincere gratitude to my thesis advisor and mentor Professor Venkatesh Raman. My association with him goes back to my undergraduate days and I have learned so many things from him that every aspect of my academic and social life is influenced by his thoughts. This thesis would not have been possible without his unstinting efforts, the benefit of his experience and his critical reviews. I would also like to thank Professor C.R. Subramanian who was my second advisor and guided me through all my graduate days and discussed with me for hours together to shape my thoughts. His contribution to this thesis is immense. One person whom I will always be in debt to is Professor Fedor V. Fomin. He not only gave me so many opportunities and guided me through my critical stage but more importantly his belief in my abilities and his unconditional support made me reach that extra mile in my academic career. I would also like to thank Professor Mike Fellows for discussing so many of his ideas and philosophies with me and also for showing me so many exciting and unexplored areas in Parameterized Complexity. All of these people have been extremely generous in giving me their valuable time and sharing me their ideas and insights. This thesis owes a debt of gratitude for their help and encouragements.

I also thank Professor Noga Alon, Professor Gregory Gutin and Professor Michael Krivelevich for fruitful collaborations and discussions in areas related to this thesis. I thank all my coauthors for allowing me to use the results obtained with them as a part of this thesis. I would also like to thank Professor Meena Mahajan, Professor V. Arvind, Professor R. Ramanujam, Professor Kamal Lodaya, Professor Madhavan Mukund, Professor K. Narayan Kumar and Professor K.V. Subrahmanyam for their help, suggestions and encouragements during my undergraduate and graduate days. I thank all the CS faculty of CMI and IMSc for the nurturing and advising they provided me over the last eight years.

I would also like to thank Somnath Sikdar, Serge Gaspers, Sounaka Mishra, Omid Amini, Rahul Muthu, Nutan P. Limaye, N. Narayanan, Daniel Lokshtanov, Yngve Villanger and Alexey Stephanov for discussing so many of their ideas and listening to my endless ramblings on various things. I will take this opportunity to thank all my undergraduate friends Amit Jayant Deshpande, Navin Rustagi, Himadri Mukherjee, Debajyoti Nandi, Yashonidhi Pandey, Suman Bandhyopadhyay, Debapriyo Majumdar, Prakash Chandrasekaran, Sourav Chakraborty and Raghav Ramesh Kulkarni for all that I know. I would especially like to thank Sushmita Gupta, a good friend, collaborator and my best critic over the past few years.

My deepest gratitude is to my family for instilling a love of knowledge in me. This thesis means as much to me as to my father, mother, grandparents and to my younger brothers Nishant and Ankit. I always feel the warmth of their love, even though we live so far apart. It is their encouragement and support that has made it possible for me to take up a career in research. This thesis is dedicated to my parents, my family and my friends.

Abstract

An important goal of the theory of algorithms is to design efficient algorithms that solve computationally difficult problems. When considering the class of NP-hard combinatorial optimization problems, the goal "appears" beyond our reach. As these problems need to be solved *exactly* for various applications in theory and practice routinely, exact algorithms become unavoidable.

Parameterized Complexity is an exact algorithmic approach to deal with intractable computational problems having some small parameters. For decision problems with input size n, and a parameter k (which typically, and in all the problems we consider in this paper, is the solution size), the goal here is to design an algorithm with runtime $f(k)n^{O(1)}$ where f is a function of k alone, against a trivial $n^{k+O(1)}$ algorithm. Problems having such an algorithm are said to be fixed parameter tractable (FPT), and such algorithms are practical when small parameters cover practical ranges.

In recent years, there has been a growing interest in designing exact algorithms for optimization versions of NP-complete problems. This has led to the development of fast exponential-time algorithms for various problems including SATISFIABILITY, GRAPH COLORING, MAXIMUM INDEPENDENT SET and many others. These exponential-time algorithms lead to practical algorithms for at least moderate instance sizes.

In this thesis we look at various problems from the exact algorithm paradigm for both parameterized and optimization versions of the problems. The first part of the thesis consists of FPT algorithms for various graph problems and the other part consists of exact algorithms for optimization versions of a few graph problems. We first give FPT algorithms for FEEDBACK VERTEX SET in undirected graphs, FEEDBACK SET Problems in Tournaments and duals of FEEDBACK SET problems in directed graphs. Then we show that several problems like DOMINATING SET, INDEPENDENT SET that are hard for various "parameterized complexity classes" on general graphs, become fixed parameter tractable on graphs with no small cycles. As our last algorithmic result we give an FPT algorithm for the DIRECTED MAXIMUM LEAF OUT-TREE problem which is the problem of finding a directed out-tree with at least k leaves in directed graphs. Finally, we give W[1]-hardness results for variations of coloring problems like LIST COLORING and EQUITABLE

Coloring when parameterized by the treewidth of the input graph.

In the second part of the thesis, we first introduce three techniques to design non-trivial exact algorithms and illustrate these techniques with several examples. Our first technique obtains a non-trivial exact algorithm for optimization versions of various problems using parameterized algorithms for the same problems. Our second technique illustrates the idea of designing exact algorithms by enumerating maximal independent sets (MIS) in a graph. We exemplify this technique by designing the currently fastest polynomial space exact algorithms for Odd Cycle Transversal, Minimum Maximal Matching, Minimum Edge Dominating Set and Matrix Dominating Set. Our last technique is based on different combinations of Branch & Reduce and dynamic programming on graphs of bounded treewidth. We illustrate this technique by giving the fastest known algorithms for a number of NP hard problems including Minimum Maximal Matching and its variations and counting the number of 3-colorings of a graph. We also apply this technique to design parameterized algorithms for various problems. Finally, we give exact algorithms for optimization versions of Maximum r-Regular Induced Subgraph problems. We give an $\mathcal{O}(c^n)$ time algorithms for these problems for any fixed constant r, where c is a positive constant strictly less than 2, depending on r alone.

Contents

List of Figures

List of Tables

Part I

Introduction

1 Introduction

Computer Science is a science of abstraction - creating the right model for a problem and devising the appropriate mechanizable techniques to solve it.

- A. Aho and J. Ullman

Classical complexity broadly divides problems into P, polynomial time solvable problems or NP, non deterministic polynomial time solvable problems. It is a widely held notion that polynomial-time computability captures feasible computation and NP-completeness identifies hard problems in this framework. One of the greatest achievements in theoretical computer science is the development of NP - completeness theory. NP-completeness theory provides a solid and convincing foundation for the study of computationally intractable problems. The importance of NP-completeness and its impact on the development of theory can be best illustrated by the following quote from the plenary lecture *P, NP and Mathematics- a computational complexity perspective*, given by Avi Wigderson at the 2006 International Congress of Mathematicians (ICM2006) [224], that took place in Madrid.

> . . . NP-completeness is a unique scientific discovery Ű there seems to be no parallel scientific notion which so pervaded so many fields of science and technology. It became a standard for hardness for problems whose difficulty we have yet no means of proving. It has been used both technically and allegorically to illustrate a difficulty or failure to understand natural objects and phenomena. Consequently, it has been

used as a justification for channeling effort in less ambitious (but more productive) directions. . . .

However, the theory does not make obsolete the pressing need for solving these hard problems because of their practical importance. The NP-hard problems cannot be wished away and have to be handled algorithmically. To emphasize this we again quote from the plenary lecture of Avi Wigderson at ICM2006 [224].

. . . The class NP is extremely rich (we shall see examples a little later). There are literally thousands of NP problems in mathematics, optimization, artificial intelligence, biology, physics, economics, industry and more which arise naturally out of different necessities, and whose efficient solutions will benefit us in numerous ways. They beg for efficient algorithms, but decades (and sometimes longer) of effort has only succeeded for a few. . . .

Many approaches have been proposed to deal with this intractability. One prominent approach to deal with NP-hard optimization problems is to settle for polynomial-time computable (good) approximate solutions. Another, more classical, approach is to identify subclasses of instances of NP-hard problems which are feasibly solvable. Both approaches have attained a reasonable degree of success [130]. Other approaches include polynomial-time randomized approximation algorithms, and heuristic algorithms. None of these approaches has satisfied all needs requested from industry and applications: polynomial-time approximation algorithms can only provide approximate solutions while certain applications may require optimal solutions; and heuristic algorithms in general do not have formal performance guarantees and often lack theoretical analysis. In short, there are many instances where one is interested in the exact solution of the problem and the instance doesn't exhibit any special structure. Anyways as Albert Einstein said:

God does not play dice with the universe.

Parameterized Complexity deals with such situations (where exact algorithm is sought for) where the instance has several parameters, some of them likely to be small for all practical purposes. Here the runtime is defined as a function of input size and the parameters. These functions are generally allowed to grow arbitrarily in parameter values but polynomially in the input size. One of the major focus of this thesis is to study parameterized algorithms for various graph problems. The other part of the thesis focuses on algorithms where the parameter is just the standard *input size*. These algorithms are generally called Exact Exponential Time Algorithms. These are basically exact algorithms for the optimization version of a problem. Both Parameterized Complexity and Exact Exponential Time Algorithms seek for exact algorithms and could be thought as paradigms looking for clever navigation through big sized *universe of solution*.

This thesis is in search of an answer to the following question asked by Avi Wigderson, in his plenary lecture at ICM2006 [224] in the realm of the above described algorithmic paradigms.

> ... problems in NP have trivial exponential time algorithms. Such algorithms search through all possible short witnesses, and try to verify each. **Can we always speed up this brute-force algorithm?** ...

This chapter is organized as follows. In the next Section 1.1, we introduce Parameterized Complexity and in Section 1.2 we introduce Exact Exponential Time Algorithms. Finally we conclude this chapter in Section 1.3, which gives the organization of the rest of the thesis.

1.1 Parameterized Complexity

Parameterized complexity is basically a two-dimensional generalization of "P vs. NP" where in addition to the overall input size n, one studies the effects on computational complexity of a secondary measurement that captures additional relevant information. This additional information can be, for example, a structural restriction on the input distribution considered, such as a bound on the treewidth of an input graph. Parameterization can be deployed in many different ways; for general background on the theory see [79, 114, 189].

There has been significant progress in obtaining practical algorithms for some problems whose inputs have often two parameters, and one of them is a very small fixed value, as illustrated below. These problems are hard (for some appropriate notion of hardness) in general, and the small parameter value covers important practical applications.

- The Simplex algorithm gives an $O(n^d)$ (assuming $d < n$) algorithm for the linear programming problem on d variables and n constraints. However Meggido [179] gave an algorithm for the problem that takes $O(2^{2^{O(d)}}n)$ time. Clearly, this will be a much better algorithm when d is very small compared to n.

- Given an undirected graph G, the achromatic number is the largest number of colors that can be assigned to the vertices of G so that adjacent vertices are assigned different colors and any two different colors are assigned to some pair of adjacent vertices. Given a graph G and an integer k, it is NP-complete to determine whether G has achromatic number at least k. However, for each fixed integer k, it can be determined in $O(|E(G)|)$ time whether or not G has achromatic number at least k [98]. (Note that such a result is not conceivable for the chromatic number problem as even for a fixed k (like 3), it is NP-complete to check whether the graph has chromatic number k.)

- Given an undirected graph G on n vertices, and an integer k, it is NP-complete [127] to determine whether G has a vertex cover of size at most k; however there is now an $O(kn + (1.2738)^k)$ [59] algorithm to answer this question.

Parameterized versions of the Dominating Set problem (a subset of vertices such that for every vertex of the graph either the vertex is in this subset or one of its neighbors is in this subset) and the Clique problem (a subset of vertices such that there is an edge between any pair of vertices of this subset), on the other hand, have only an $\Omega(n^k)$ algorithm where k is the fixed parameter and n is the size of the input. Parameterized complexity, pioneered by Downey and Fellows [74, 79] is a systematic attempt to study this contrasting behaviour of the role of the parameter in fixed parameter problems. Through this, we systematically look at some algorithmic techniques that have been developed to prove problems fixed parameter tractable. There is also a completeness theory known in this framework to prove apparently fixed parameter intractable problems.

A *parameterized problem* is a set $L \subseteq \Sigma^* \times \Sigma^*$ where Σ is a fixed alphabet. For convenience, we consider that a parameterized problem L is a subset $L \subseteq \Sigma^* \times N$. For a parameterized problem L and $k \in N$ we write L_k to denote the associated fixed-parameter problem $L_k = \{x|(x, k) \in L\}$.

Decision versions of most NP-complete problems have natural parameters.

Definition 1.1 ([74]) *A parameterized problem L is said to be (uniformly) fixed-parameter tractable (FPT) if there is a constant α and an algorithm Φ such that Φ decides if $(x, k) \in L$ in time $f(k)|x|^\alpha$ where $f : N \to N$ is an arbitrary function.*

Let FPT denote the class of all fixed-parameter tractable parameterized problems. We will also call an algorithm that places a parameterized problem in FPT a fixed parameter algorithm. Just as 'polynomial time' usually refers to time a polynomial in the input size, by 'fixed parameter time' we refer to time $O(f(k))$ times polynomial in the input size where f is an arbitrary function of k, the parameter.

Under this notion, the Linear programming where the dimension is the parameter, the k-achromatic number problem, k-vertex cover and the parameterized versions of several well known NP-complete problems [79] are fixed parameter tractable. In fact, the Linear Programming example illustrates the fact that this theory also addresses problems other than parameterized versions of NP-complete problems.

There is a hierarchy of intractable parameterized problem classes above FPT, the main ones are:

$$FPT \subseteq M[1] \subseteq W[1] \subseteq M[2] \subseteq W[2] \subseteq \cdots \subseteq W[P] \subseteq XP.$$

The principal analogue of the classical intractability class NP is $W[1]$. A convenient source of $W[1]$-hardness reductions is provided by the result that INDEPENDENT SET is complete for $W[1]$ [79]. Other highlights of the theory include that DOMINATING SET, by contrast, is complete for $W[2]$ [79].

We give a more formal treatment of some of these complexity classes in the next chapter.

1.2 Exact Exponential Time Algorithms

Every problem in NP can be solved in exponential time by an exhaustive search. Recall that a decision problem is in NP, if and only if there exists a polynomial time decidable relation $R(x, y)$ and a polynomial $m(|x|)$ such that for every yes-instance x, there exists a yes-certificate y with $|y| \leq m(|x|)$ and $R(x, y)$. A trivial exact algorithm for solving instance x enumerates all possible strings y of length up to $m(|x|)$, and checks whether any of them yields a yes-certificate. Up to polynomial factors that depend on the evaluation time of $R(x, y)$, this yields an exponential running time of $2^{m(x)}$.

A natural question is: Can we do better than this trivial enumerative algorithm? Interestingly, for many combinatorial optimization problems the answer is YES. The design of exact algorithms has a long history dating back to Held and Karp's paper [145] on the TRAVELING SALESMAN problem in the early sixties. Other early examples include an $O^*(1.4422^n)$ algorithm for deciding 3-colorability of an n-vertex graph by Lawler [170], an $O^*(1.2599^n)$ algorithm for finding a maximum independent set in an n-vertex graph by Tarjan and Trojanowski [217], an $O^*(1.4142^n)$ algorithm for the SUBSET SUM problem with n integers by Horowitz and Sahni [146]. In recent years, there has been a growing interest in designing exact algorithms for NP-complete problems. This has led to the development of fast exponential time algorithms for various problems including SATISFIABILITY [133, 150, 225], GRAPH COLORING [88, 41], MAXIMUM INDEPENDENT SET [208], MAX CUT [225], and many others. These exponential-time algorithms lead to practical algorithms for at least moderate instance sizes. Furthermore, there is a wide variation in the time complexities of exact algorithms for NP-complete problems. Classical complexity theory cannot explain these differences. The study of exact algorithms may lead to a finer classification, and hopefully a better understanding, of NP-complete problems. See the recent surveys [117, 212, 226, 227] for an overview and recent developments in the area.

1.3 Organization of the rest of the Thesis

This thesis is organized as follows. It is broadly divided into four parts:

- Introduction,
- Fixed Parameter Tractable Algorithms and Intractability,
- Exact Exponential Time Algorithms and
- Conclusion and Future Directions.

In Chapter 2, we give necessary definitions, set up other notations and outline a few algorithmic techniques that will be used throughout the thesis.

The second part of thesis primarily contains fixed parameter tractable algorithms for various graph problems except for the last chapter. In the last chapter of the second part we give hardness proofs for a few variations of coloring problems. The third part of the thesis contains non-trivial exact exponential time algorithms for optimization version of graph problems. The final part of the thesis contains an up-to-date summary on the algorithmic complexity of the problems considered in the thesis and gives possible future directions to explore.

Chapter 3 deals with FPT algorithms for the Feedback Vertex Set problem in an undirected graph while Chapter 4 gives FPT algorithm for feedback set problems in a special class of directed graphs, tournaments. In Chapter 5 we give FPT algorithms for many basic problems like Independent Set and Dominating Set in graphs which do not have cycles of length 3 and 4 as subgraph. Chapter 6 gives FPT algorithms for finding a directed tree with at least k leaves. We conclude this part in Chapter 7 by showing that a few variations of coloring problems like list coloring and equitable coloring are W[1]-hard even on graphs of bounded treewidth.

In the third part of the thesis we first give three new techniques to devise exact exponential time algorithms. In Chapter 8, we show how to obtain exact algorithms by using parameterized versions and enumeration of subsets. Chapter 9 gives various exact algorithms based on enumeration of maximal independent sets. In Chapter 10, we devise a technique which is based on different combinations of treewidth and branching and give several applications of it, including an algorithm for Minimum Maximal Matching and counting 3 coloring. Chapter 11 gives an algorithm to find a maximum r-regular induced subgraphs for any constant r.

We conclude with a summary, some remarks, discussions and open problems in Chapter 12 of the final part of the thesis.

2 Preliminaries

In this chapter we set up all the notations, give definitions and overview some of the techniques used in the thesis.

We start with defining different decompositions of graphs which are central for many of the algorithms presented in the thesis.

Section 2.2 gives us the notations used to denote the time complexity of the algorithms. In Section 2.3 we give a method to solve recurrence relations which will be used heavily in the analysis of various algorithms developed in this thesis

Just like the divide and conquer, dynamic programming and DFS/BFS search techniques for general algorithm design, several techniques have emerged during the last decade to show a problem fixed parameter tractable. Some of these techniques are elementary but powerful while others are based on the deep Robertson-Seymour graph minor theorems, bounded treewidth machinery and so on. In Section 2.4 we illustrate some of these techniques through several examples. There is also a theory of parameterized intractability using which one can identify parameterized problems that are unlikely to admit parameterized algorithms. In Section 2.5 we briefly look at the parameterized intractability theory, which includes various complexity classes and parameterized reductions.

In Section 2.6 we give an algorithmic technique namely *Measure and Conquer*, which is best suited for designing Exact Exponential Time algorithms.

2.1 Different Decompositions of Graphs

One of the major mathematical results of recent times is *Graph Minor Theorem* of Robertson and Seymour. On the way to these results they have introduced various methods to decompose graphs and many width techniques like *Treewidth* and *Pathwidth* have been introduced. We first define *contraction* and then define different width techniques.

The *contraction* of an edge of a graph, also called edge contraction, is the graph obtained by identifying both end points of an edge to a single vertex which is made adjacent to all the vertices that were adjacent to either of the end points of the edge.

Definition 2.1 *Let $G = (V, E)$ be a graph. A tree decomposition of G is a pair $(\{X_i | i \in I\}, T = (I, F))$ with $\{X_i | i \in I\}$ a family of subsets of V, and T a tree, with the following properties:*

- $\bigcup_{i \in I} X_i = V$
- *For every edge $e = (v, w) \in E$, there is an $i \in I$ with $v \in X_i$ and $w \in X_i$.*
- *For every $v \in V$, the set $\{i | v \in X_i\}$ forms a connected subtree of T.*

The *treewidth* of a tree decomposition $(\{X_i | i \in I\}, T)$ is $max_{\{i \in I\}}(|X_i| - 1)$. The *treewidth* of G is the minimum treewidth, taken over all possible tree decompositions of G.

If in the definitions of a tree decomposition and treewidth we restrict T to be a path then we have the definitions of *path decomposition* and *pathwidth*. We use the notation $\mathbf{tw}(G)$ and $\mathbf{pw}(G)$ to denote the treewidth and the pathwidth of a graph G.

2.2 Notations Related to Time Complexity

In the thesis two commonly used notations to represent the time complexity of an algorithm are $\mathcal{O}$ and $\mathcal{O}^*$. Formally, suppose $f(n)$ is a function defined on natural numbers that is $f : \mathbb{N} \rightarrow \mathbb{N}$, then we write $f(n) = \mathcal{O}(g(n))$ if there exists constants c and n_0 such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$. Our other notation $\mathcal{O}^*$ is primarily meant for exponential time algorithms. The $\mathcal{O}^*$ notation was introduced by Woeginger in [226]. Here we write $\mathcal{O}^*(T(x))$ for a time complexity of the form $\mathcal{O}(T(x)|x|^c)$, where c is some constant. One could also use this notation for a parameterized algorithm by focusing on function which depends on the parameters alone. Thus we write $\mathcal{O}^*(f(k))$ for the time complexity of the form $\mathcal{O}(f(k)|x|^c)$ where k is a parameter value, $|x|$ is the size of the input and c is some constant.

2.3 A Technique to Solve Recurrence Relations

Many of the algorithms in this thesis are based on the "bounded search tree technique" and the running time of the algorithms are estimated using the number of nodes in the search tree. This number is computed using recurrence relations. The recurrence relations we obtain are almost always linear, with constant coefficients and there exist several well-known techniques for solving them. Suppose the algorithm solves a problem of size n by calling itself recursively on problems of sizes $n-d_1, n-d_2, \cdots, n-d_p$, then $(d_1, d_2, \cdots, d_p)$ is called the *branching vector* of this recursion. Then the runtime is obtained using the recurrence

$$T(n) = T(n-d_1) + T(n-d_2) + \cdots + T(n-d_p) + O(n^c). \tag{2.1}$$

Here, the final term of $O(n^c)$ is the time spent besides the recursive calls (to break the problem into subproblems and/or to obtain the overall solution from the solution of the subproblems). The characteristic polynomial of this recurrence is given by

$$x^d = x^{d-d_1} + x^{d-d_2} + \cdots + x^{d-d_p}. \tag{2.2}$$

where d is maximum of $\{d_1, d_2 \cdots, d_p\}$. If γ is the unique positive root of Equation 2.2 then $T(n)$ is $\mathcal{O}^*(\gamma^n)$. We call γ the *branching number* that corresponds to the branching vector $(d_1, d_2, \cdots, d_p)$.

2.4 Techniques to Prove FPT

Demonstrations of fixed-parameter tractability sometimes uses novel approaches that shift the complexity burden onto the parameter. Some of these approaches run counter to our established practices of thought in designing polynomial time algorithms. In the parameterized setting, as Downey and Fellows [75] say, 'the parameter can be "sacrificed" in interesting ways'. In this section we give a few techniques which is useful in showing a problem to be fixed parameter tractable.

2.4.1 Reduction to Kernel

The idea of this method is to reduce the given problem instance in fixed parameter time to an instance whose size is bounded by some function of the parameter. Then the new instance is exhaustively analyzed to find the solution. More precisely, the main idea of *kernelization* is to replace a given instance (I, k) by a simpler instance (I', k') using some *data reduction rules* in FPT time (often in polynomial time) such that (I, k) is a yes-instance if and only if (I', k') is a yes-instance and $|I'|$ is bounded by a function of k alone. The reduced instance is called *kernel* for the problem. Now we could apply any brute force algorithm on this kernel to solve the original problem. We illustrate the technique using the parameterized version of MAX3SAT where given a boolean 3-CNF formula and an integer parameter k, we would like to know whether there is an assignment to the variables that satisfies at least k of the clauses. Our other examples in this Section include a kernel for d-HITTING SET using the *Sunflower Lemma* and a $4k$ sized kernel for VERTEX COVER using *crown decomposition.*

Max3Sat

Let F be a given boolean CNF 3-SAT formula with n variables and m clauses. t is well known that in any boolean CNF formula, there is an assignment that satisfies at least half of the clauses (given any assignment that doesn't satisfy half the clauses, its bitwise complement will). So if the parameter k is less than $m/2$, then there is an assignment to the variables that satisfies at least k of the clauses. Otherwise, $m \leq 2k$, and so $n \leq 6k$. Now, we can exhaustively try all the 2^n which is at most 2^{6k} assignments to test whether at least k of the clauses can be satisfied.

Thus we have reduced the given formula trivially so that n, the number of variables and m, the number of clauses become functions of k after which we apply a brute force technique. (See [42] and [177] for parameterized algorithm for general versions of MaxSat and other parameterizations.)

d-Hitting Set

In this Section we give a kernelization algorithm for the d-Hitting Set problem which is defined as follows:

> d-Hitting Set (d-HS) : Given a collection $\mathcal{C}$ of d element subsets of an universe U and a positive integer k, the problem is to determine whether there exists a subset $U' \subseteq U$ of size at most k such that U' contains at least one element from each set in $\mathcal{C}$.

Our kernelization algorithm is based on the following widely used *Sunflower Lemma*. We first define the terminology used in the statement of the lemma. A *sunflower* with k *petals* and a *core* Y is a collection of sets $S_1, S_2 \cdots S_k$ such that $S_i \cap S_j = Y$ for all $i \neq j$; the sets $S_i - Y$ are petals and we require that none of them be empty. Note that a family of pairwise disjoint sets is a sunflower (with an empty core).

Lemma 2.1 ([155]) [Sunflower Lemma] *Let $\mathscr{F}$ be a family of sets over an universe $\mathscr{U}$ each of cardinality s. If $|\mathscr{F}| > s!(k-1)^s$ then $\mathscr{F}$ contains a sunflower with k petals and such a sunflower can be computed in time polynomial in the size of $\mathscr{F}$ and $\mathscr{U}$.*

Now we are ready to prove the following theorem about kernelization for d-HS.

Theorem 2.1 *d-HS has a kernel of size $O(k^d d! d^2)$. That is, given an instance $(U, \mathcal{C}, k)$ of d-HS, we can replace it with an equivalent instance $(U, \mathcal{C}', k')$ with $|\mathcal{C}'| \leq O(k^d d! d)$ in polynomial time.*

Proof: The crucial observation is that if $\mathcal{C}$ contains a sunflower $S = \{S_1, \cdots, S_{k+1}\}$ of cardinality $k+1$ then every hitting set of $\mathcal{C}$ of size at most k must intersect with the core Y of the sunflower S, otherwise we will need hitting set of size more than k. Therefore if we let $\mathcal{C}' = \mathcal{C} \setminus (S \cup Y)$ then the instance $(U, \mathcal{C}, k)$ and $(U, \mathcal{C}', k)$ are equivalent.

Now we apply the Sunflower Lemma for all $d' \in \{1, \cdots, d\}$, repeatedly replacing sunflowers of size at least $k + 1$ with their cores until the number of sets for any fixed $d' \in \{1, \cdots, d\}$ is at most $O(k^{d'} d'!)$. Summing over all d we obtain the desired kernel of size $O(k^d d! d)$. □

Crown Decomposition : Vertex Cover

In this Section we introduce a *crown decomposition* based kernelization for Vertex Cover. It is based on a connection between matchings and vertex cover which is that the maximum size of a matching is a lower bound for the minimum cardinality vertex cover. We first define Vertex Cover precisely as follows.

> Vertex Cover (VC): Given a graph $G = (V, E)$ and a positive integer k, does there exist a subset $V' \subseteq V$ of size at most k such that for every edge $(u, v) \in E$ either $u \in V'$ or $v \in V'$.

Vertex Cover can be modelled as 2-HS with universe $U = V$ and $\mathcal{C} = \{\{u, v\} \mid (uv) \in E\}$ and hence using Theorem 2.1 we get a kernel with at most $4k^2$ edges and $8k^2$ vertices. Here we give a kernel with at most $4k$ vertices.

Now we define crown decomposition.

Definition 2.2 *A crown decomposition of a graph $G = (V, E)$ is a partitioning of V as C, H and R, where C and H are nonempty and the partition satisfies the following properties.*

1. *C is an independent set.*
2. *There are no edges between vertices of C and R, that is $N[C] \cap R = \emptyset$.*
3. *Let E' be the set of edges between vertices of C and H. Then E' contains a matching of size $|H|$, that is the bipartite subgraph $G' = (C \cup H, E')$ has a matching saturating all the vertices of H.*

We need the following lemma by Chor et. al. [60] which makes it possible to find a crown decomposition efficiently.

Lemma 2.2 *If a graph $G = (V, E)$ has an independent set $I \subseteq V$ such that $|N(I)| < |I|$, then a crown decomposition (C, H, R) of G such that $C \subseteq I$ can be found in time $O(m + n)$, given G and I.*

The crown-decomposition gives us a global method to reduce the instance size. Its importance is evident from the following simple lemma.

Lemma 2.3 *Let (C, H, R) be a crown decomposition of a graph $G = (V, E)$. Then G has a vertex cover of size k if and only if $G' = G[R]$ has a vertex cover of size $k' = k - |H|$.*

Proof: Suppose G has a vertex cover V' of size k in G. Now, we have a matching of size $|H|$ between C and H that saturates every vertex of H. Thus $|V' \cap (H \cup C)| \geq |H|$, as any vertex cover must pick one vertex form each of the matching edge. Hence the number of vertices in V' covering the edges not incident to $H \cup C$ is at most $k - |H|$, proving one direction of the result.

For the other direction, it is enough to observe that if V'' is a vertex cover of size $k - |H|$ for G' then $V'' \cup H$ is a vertex cover of size k for G. $\square$

Theorem 2.2 *Vertex Cover has a kernel of size $4k$.*

Proof: Given an input graph $G = (V, E)$ and a positive integer k, we do as follows. We first find a maximal matching M of G. Let $V(M)$ be the set of endpoints of edges in M. Now if $|V(M)| > 2k$, we answer NO and stop as any vertex cover must contain at least one vertex from each of the matching edges and hence has size more than k. Now we distinguish two cases based on the size of $|V - V(M)|$. If $|V - V(M)| \leq 2k$, then we stop as we have obtained a kernel of size at most $4k$. Else $|V - V(M)| > 2k$. In this case we have found an independent set $I = V - V(M)$ such that $|N(I)| \leq |V(M)| < |I|$ and hence we can apply Lemma 2.2 to obtain a crown decomposition (C, H, R) of G. Given a crown decomposition (C, H, R), we apply Lemma 2.3 and obtain a smaller instance for a vertex cover with $G' = G[R]$ and parameter $k' = k - |H|$. Now we repeat the above procedure with this reduced instance until either we get a NO answer or we have $|V - V(M)| \leq 2k$ resulting in a kernel of size $4k$. $\square$

The bound obtained on the kernel for Vertex Cover in Theorem 2.2 can be further improved to $2k$ with much more sophisticated use of crown decomposition. An alternate method to obtain a $2k$ size kernel for Vertex Cover is through a Linear Programming formulation of Vertex Cover. See [114] and [189] for further details on Linear Programming based kernelization of Vertex Cover.

2.4.2 Bounded Search Trees

The idea here is to first identify, in polynomial time, a small (typically a constant, but even logarithmically many is also fine) subset of elements of which at least one (or a subset of smaller size) must be in *any* feasible solution of the problem. Then we include one of them at a time and recursively solve the remaining problem with the parameter value reduced by 1. If we unravel the recursion tree, the depth of this tree will be bounded by k with the branching factor bounded by the size of the small subset identified. This will bound the total size of the tree as a function of k and as in each node a polynomial time is spent, the resulting algorithm is a fixed parameter algorithm. We illustrate this technique with Vertex Cover problem.

Vertex Cover

Let $G = (V, E)$ be the input to the Vertex Cover and k be the parameter. Our algorithm is based on following two simple observations.

- For a vertex v, any vertex cover must contain either v or all of its neighbors $N(v)$.
- Vertex Cover can be solved optimally in polynomial time when the maximum degree of a graph is at most 2.

So our algorithm recursively solves the problem by finding a vertex v of maximum degree in the graph and if $d(v) \geq 3$ then recursively branching on two cases by considering either v in the vertex cover or $N(v)$ in the vertex cover. When we consider two cases like this, we say we *branch according to v and $N(v)$*. And when the maximum degree of the graph is 2, we solve the problem in polynomial time.

The time complexity of the algorithm can be described by the following recurrence in k.

$$T(k) = \begin{cases} T(k-1) + T(k-3) + n^{O(1)} & \text{if } k \geq 2 \\ n^{O(1)} & \text{if } k \leq 1. \end{cases}$$

The above recursive function bounds the size of the search tree and the time spent at each node in the tree. The above recursive function can be solved by finding the largest root of the characteristic polynomial $\lambda^k = \lambda^{k-1} + \lambda^{k-3}$. Using standard mathematical softwares the root is estimated to 1.466. This gives us the following theorem.

Theorem 2.3 *Parameterized* VERTEX COVER *can be solved in* $\mathcal{O}(1.466^k n^{O(1)})$.

2.4.3 Iterative Compression

Iterative Compression is a recently developed technique to show a problem to be fixed parameter tractable. This technique was first introduced by Reed et. al. [202] to solve the ODD CYCLE TRANSVERSAL problem, where one is interested in finding a set of at most k vertices whose deletion makes the graph bipartite. It is a useful technique for designing FPT algorithms for minimization problems. The idea here is to design a fixed parameter tractable algorithm which, given a $k+1$ sized solution for a problem, either compresses it to a solution of size k or proves that there is no k size solution. This is known as the compression step of the algorithm. Based on this compression step, recursive algorithms for minimization problems are obtained. The most technical part of algorithms based on *iterative compression* is to show that the compression step can be carried out in time $O(f(k)n^{O(1)})$. We illustrate this technique with the d-HITTING SET, problem defined in Section 2.4.1.

Let the sets in collection $\mathfrak{C}$ be $\{C_1, C_2, \cdots C_m\}$. By $\mathfrak{C}_i$ we represent the sub collection of $\mathfrak{C}$ consisting of C_j, $1 \leq j \leq i$.

Our first observation is that if there does not exist a $U' \subseteq U$, a hitting set, of size at most k for $\mathfrak{C}_i$ for any $1 \leq i \leq m$ then the answer to the whole problem is NO. That is, there does not exist a hitting set of size at most k for $\mathfrak{C}$. Now we define the compression version of d-HS problem.

> COMP-d-HITTING SET : Given a collection $\mathfrak{C}$ of subsets of size at most d of an universe U, a positive integer k and a hitting set $U' \subseteq U$ of size $k+1$ for $\mathfrak{C}$. The problem is to determine whether there exists a hitting set $\widehat{U} \subseteq U$ of size at most k for $\mathfrak{C}$.

Now we show that if we can solve COMP-d-HITTING SET problem in $f(k) \cdot m^{O(1)}$ time, where m is the cardinality of the collection $\mathfrak{C}$, then we can solve the d-HITTING SET problem in $O(f(k) \cdot m^{O(1)})$ time. The idea is to solve the problem by applying algorithm for COMP-d-HITTING SET m times. We start with $\mathfrak{C}_{k+1}$, where a solution U' of size $k+1$ can be obtained by picking arbitrarily one element from each of C_j, $1 \leq j \leq k+1$. This gives us an instance for Comp-d-Hitting Set problem. By applying an algorithm for Comp-d-Hitting Set, either we get a NO answer for $\mathfrak{C}$ or a compressed solution $\widehat{U}$ of size at most k for $\mathfrak{C}_{k+1}$. If we get a compressed solution then we move on to the instance $\mathfrak{C}_{k+2}$. Now by adding an element from $C_{k+2} = \mathfrak{C}_{k+2} - \mathfrak{C}_{k+1}$ to $\widehat{U}$, we get a U' a solution of size at most $k+1$ for $\mathcal{C}_{k+2}$ and also an instance for Comp-d-Hitting Set. So we apply an algorithm for Comp-d-Hitting Set in an orderly fashion on $\mathfrak{C}_j$ for $k+2 \leq j \leq m$ by obtaining a solution of size $k+1$ for $\mathfrak{C}_j$ by adding an element from C_j to a solution of size k for $\mathfrak{C}_{j-1}$. So on the way either we answer NO or we find a hitting set of size k for $\mathfrak{C}$.

Now we are left with solving the Comp-d-Hitting Set problem. We obtain an algorithm for Comp-d-Hitting Set problem using an algorithm for $(d-1)$-Hitting Set. For ease of presentation we do it for 3-Hitting Set problem. We later explain how to make this algorithm work for larger values of $d > 3$.

So now assume that we have an instance of Comp-3-Hitting Set problem. Given a U', we partition this set into two parts $U' = U'_{accept} \cup U'_{reject}$ (size of $|U'_{accept}| \leq k$) and for each partition we look for solution of size k containing every element of U'_{accept} and not containing any element of U'_{reject}. Given a partition $U' = U'_{accept} \cup U'_{reject}$, we include every vertex of U'_{accept} in $\widehat{U}$ (a possible solution of size k) and then apply some reduction rules on the input. These reduction rules either return NO, in which case there is no valid solution respecting this partition and hence we try an alternate partition or outputs an instance of 2-Hitting Set. More precisely our reduction rules are as follows.

(R0) If $C_j \cap U'_{accept} \neq \emptyset$ then $\mathfrak{C} = \mathfrak{C} - C_j$. That is, remove all sets from $\mathfrak{C}$ which are already hit by U'_{accept}.

(R1) If there exists a set $C_i \in \mathfrak{C}$ such that $C_i \subseteq U'_{reject}$ then answer NO and reject.

(R2) After we have exhaustively applied reduction rules $(R0)$ and $R(1)$, we obtain a new $\mathfrak{C}'$ as follows
$$\mathfrak{C}' = \{C - U'_{reject} \mid C \in \mathfrak{C}\}.$$

The soundness of the above reduction rules is obvious. Note that the reduction rules are not particular to 3-Hitting Set and can be applied to any arbitrary

d. Notice that in our reduction rules we have either removed a set hit by U'_{accept} or removed an element of a set which is in U'_{reject} and hence in the instance $\mathfrak{C}'$ obtained after reduction rules has sets of size at most 2 (in case of arbitrary d it will have sets of size $d-1$). So here $\mathfrak{C}'$ is an instance of 2-Hitting Set problem or Vertex Cover. We need the following best known result for Vertex Cover problem as a subroutine to solve d-HS.

Theorem 2.4 ([59]) Vertex Cover *can be solved in* $O(1.2738^k + kn)$ *time.*

So to solve Comp-3-Hitting Set problem, we partition U' as $U' = U'_{accept} \cup U'_{reject}$ where $|U'_{accept}| \leq k$, and for each partition apply reduction rules $(R0) - (R2)$ and obtain an instance for 2-Hitting Set problem and then apply Theorem 2.4 with parameter $k - |U'_{accept}|$. This gives us the following upper bound on the time complexity for Comp-3-Hitting Set:

$$\sum_{i=1}^{k+1} \binom{k+1}{i} O((1.2738)^{k-i} + (k-i)m) \cdot m = O(2.2738^k m + km^2).$$

This implies that we can solve 3-Hitting Set problem in $O(2.2738^k m^2 + m^3)$ time.

Theorem 2.5 3-Hitting Set *can be solved in* $O(2.2738^k m^2 + km^3)$ *time.*

Now to solve d-Hitting Set for larger values of d, we can use an algorithm for $(d-1)$-Hitting Set as a subroutine, which results in the following theorem.

Theorem 2.6 d-Hitting Set *can be solved in* $\mathcal{O}^*((d-2+1.2738)^k)$ *time.*

2.4.4 Bounded Treewidth Machinery

The notion of treewidth which is a measure of a graph to indicate how tree-like the graph is, was introduced by Robertson and Seymour [205]. We already gave a formal definition for *treewidth* in Section 2.1 but there are several alternative ways to characterize the class of graphs with treewidth $\leq k$, e.g., as partial k-trees [218] A large number of NP-complete (and other) graph problems can be solved in polynomial and even linear time when restricted to graphs with constant treewidth [14, 15, 30] if the tree decomposition is given.

The Treewidth concept has been used to prove some parameterized problems fixed parameter tractable in the following way. For these parameterized problems, fixing the parameter k implies that the yes-instances (or sometimes the no-instances) have treewidth bounded by a function of k. For a fixed k, there is a linear time algorithm due to Bodlaender [29] to test whether a given graph has treewidth at most k (the constant of proportionality is exponential in k^2). We first use this algorithm for such problems to identify whether the treewidth of the given graph is that bounded function of k (otherwise it will be a no or yes instance appropriately) and if so we get the tree decomposition as well from Bodlaender's algorithm. Given the tree decomposition, we apply the polynomial time algorithm for the problem for bounded treewidth graphs, thus obtaining a fixed parameter tractable algorithm.

We illustrate this technique through the following parameterized problem: Given an undirected graph G, and an integer parameter k, does G have a cycle of length at least k? We first need the following result.

Theorem 2.7 ([15, 27]) *There exists an $O(2^k k! n)$ algorithm to find the longest cycle (or longest path) in a given graph G together given a tree decomposition of G that has treewidth at most k.*

To find whether a given graph G has a cycle of length at least k, first grow a depth first tree rooted at any vertex noting the depth first number for each vertex. When a back edge is encountered, if the difference between the two dfs numbers is at least $k-1$, we have already encountered a cycle of length at least k. Otherwise, once the DFS tree T is constructed, for all $v \in V$, let X_v be the set containing v and its at most $k-2$ direct predecessors in T. Then $(\{X_v | v \in V\}, T)$ is a T-based tree decomposition of G of treewidth at most $k-2$. Thus (by using Bodlaender's algorithm or by the DFS tree method above) we can find a cycle of length at least k or find a tree decomposition of width at most $k-2$. In the latter case, we apply the Theorem 2.7 to test for the existence of a cycle of length k or more.

A similar algorithm can be used to find a path of length at least k. This algorithm is due to Bodlaender [27]. There is an $O(nm)$ algorithm to find cycles of length exactly k even in directed graphs by Monien [187].

2.5 Fixed Parameter Intractability

Parameterized Complexity is rich not only because of fixed parameter tractable algorithms but also because of its wide negative tool kit which allows us to show *impossibility results* that a problem is not FPT. The importance of intractability theory is best seen in view of the following quote from Hilbert's 1900 lecture.

> Proofs of impossibility were effected by the ancients ... [and] in later mathematics, the question as to the impossibility of certain solutions plays a preeminent part. ...
>
> In other sciences also one meets old problems which have been settled in a manner most satisfactory and most useful to science by the proof of their impossibility. ... After seeking in vain for the construction of a perpetual motion machine, the relations were investigated which must subsist between the forces of nature if such a machine is to be impossible; and this inverted question led to the discovery of the law of the conservation of energy. ...
>
> It is probably this important fact along with other philosophical reasons that gives rise to conviction ... that every definite mathematical problem must necessary be susceptible of an exact settlement, either in the form of an actual answer to the question asked, or by the proof of the impossibility of its solution and therewith the necessary failure of all attempts. ... This conviction ... is a powerful incentive to the worker. We hear within us the perpetual call: There is the problem. Seek its solution. You can find it by pure reason, for in mathematics there is no ignorabimus.

In the following Section we first define parameterized reductions and then various parameterized complexity classes.

2.5.1 Problem Reductions

We start this Section with the following paragraph from Flum and Grohe [114], which best describes the essence of parameterized reduction.

"Algorithms have always been analyzed and optimized in terms of many different input parameters, and no complexity theory was needed to do this. The main contribution of the theory is to provide a framework for establishing the intractability of certain problems. In the absence of techniques for actually proving lower bounds for natural problems, the main goal of such theory is to classify problems into complexity classes by means of suitable reductions. Efficient reductions are the backbone of computational complexity. Since the parameterized theory is two-dimensional, depending not only on the input size but also on the parameter, it is not surprising that it leads to a much larger variety of complexity classes and to more complicated reductions than the classical, one dimensional complexity theory."

A parameterized reduction is defined as follows.

Definition 2.3 ([74]) *Let A, B be parameterized problems. We say that A is (uniformly many:1) reducible to B if there is an algorithm Φ which transforms (x, k) into $(x', g(k))$ in time $f(k)|x|^{\alpha}$, where $f, g : N \to N$ are arbitrary functions and α is a constant independent of k, so that $(x, k) \in A$ if and only if $(x', g(k)) \in B$.*

It is easy to see that if A reduces to B and B is fixed parameter tractable then so too is A. It is important to note that there are two ways in which parameterized reductions differ from familiar P-time reductions: (1) the time taken for reduction may be polynomial in n, but (for example) exponential in the parameter k, and (2) the parameter k must map to $g(k)$ (unlike NP-completeness reductions which may map k to $k' = n - k$, for example).

2.5.2 Complexity Classes

The classes are intuitively based on the complexity of the circuits required to check a solution, or alternatively, the "natural logical depth" of the problem.

Definition 2.4 ([74]) *A Boolean circuit is of* mixed type *if it consists of circuits having gates of the following kinds.*
(1) **Small gates***:* **not** *gates,* **and** *gates and* **or** *gates with bounded fan-in. We will usually assume that the bound on fan-in is 2 for* **and** *gates and* **or** *gates, and 1 for* **not** *gates.*
(2) Large gates: **and** *gates and* **or** *gates with unrestricted fan-in.*

Definition 2.5 ([74]) *The depth of a circuit C is defined to be the maximum number of gates (small or large) on an input-output path in C. The weft of a circuit C is the maximum number of large gates on an input-output path in C.*

Definition 2.6 ([74]) *We say that a family of decision circuits F has bounded depth if there is a constant h such that every circuit in the family F has depth at most h. We say that F has bounded weft if there is constant t such that every circuit in the family F has weft at most t. The weight of a boolean vector x is the number of 1's in the vector.*

Definition 2.7 ([74]) *Let F be a family of decision circuits. We allow that F may have many different circuits with a given number of inputs. To F we associate the parameterized circuit problem $L_F = \{(C, k) : C$ accepts an input vector of weight $k\}$.*

Definition 2.8 ([74]) *A parameterized problem L belongs to $W[t]$ if L reduces to the parameterized circuit problem $L_{F(t,h)}$ for the family $F(t, h)$ of mixed type decision circuits of weft at most t, and depth at most h, for some constant h.*

For example, the parameterized clique problem is in $W[1]$ because a given graph $G(V, E)$ has a clique of size k if and only if the boolean expression $\wedge_{(i,j)\notin E}(\bar{x}_i \vee \bar{x}_j)$ has a weight k satisfying assignment. Now, using the above expression, one can reduce the clique problem to a parameterized weft 1, depth 2 circuit problem.

Definition 2.9 *A parameterized problem L belongs to $W[P]$ if L reduces to the circuit problem L_F, where F is the set of all circuits (no restrictions on weft/depth).*

The framework above describes a hierarchy of parameterized complexity classes

$$FPT \subseteq W[1] \subseteq W[2] \subseteq \cdots \subseteq W[P]$$

for which there are many natural hard or complete problems [76]. The union of these classes has been termed the W Hierarchy, denoted by WH. It is easy to see that if $P = NP$ then $WH \subseteq FPT$.

A parameterized variation of Satisfiability based on a normal form for boolean expressions has been shown to be a canonical complete problem for the various levels of WH [79].

Definition 2.10 *A boolean expression X is termed t-normalized if:*

1. *$t = 2$ and X is in product-of-sums (P-o-S) form,*
2. *$t = 3$ and X is in product-of-sums-of-products (P-o-S-o-P) form, and*
3. *for general t, X is in product-of-sums-of.... form with t alterations.*

Thus, according to the definition of 2.10, 2-normalized is the same as CNF. The *weighted t-Normalized Satisfiability problem is:* given a t-normalized boolean expression X, and a positive integer parameter k, does X have a satisfying truth assignment of weight k?

Theorem 2.8 ([76]) *Weighted t-Normalized Satisfiability is complete for $W[t]$ for $t \geq 2$.*

Furthermore, all of the following problems are now known to be complete for $W[1]$: Square tiling, Independent set, Clique, Bounded post correspondence problem, k-Step derivation for context-sensitive grammars, Vapnik-Chervonenkis dimension, and the k-Step halting problem for nondeterministic Turing machines [43, 73, 80]. Thus, any one of these problems is fixed-parameter tractable if and only if all of the others are; and none of the problems for which we know W hardness results is fixed-parameter tractable unless all of these are also. Dominating set is complete for $W[2]$ [74]. Hence fixed parameter tractability for Dominating set, or any other $W[2]$-hard problem implies fixed parameter tractability for all problems in $W[1]$ mentioned above, and all other problems in $W[2] \supseteq W[1]$.

2.6 Measure and Conquer with Branch and Reduce

Measure and Conquer is basically a tool to analyze exact exponential time algorithms. The technique when applied to algorithms based on bounded search tree of Section 2.4.2 yields amazing results. In the framework of exact exponential time algorithms, bounded search tree method is called as Branch and Reduce, as here we reduce the size of the input rather than the parameter during the recursive calls made by the algorithm. We illustrate the method with an exact algorithm for optimization version of Independent Set which we call Maximum Independent Set. The problem is defined as follows.

> Maximum Independent Set (MIS): Given a graph $G = (V, E)$, find a maximum sized subset $V' \subseteq V$ such that $G[V']$ is an independent set that is it does not have any edge.

Independent Set is the complement of Vertex Cover and hence for the optimization version, both MIS and Minimum Vertex Cover problem are equivalent. In Section 2.4.2, we gave an algorithm for the parameterized version of vertex cover problem. The goal here is for an optimum solution and not a k sized solution.

Our exact algorithm for MIS is based on the following observations:

- For a vertex v, any independent set either contains v and none of its neighbors or does not contain v.
- MIS can be solved optimally in polynomial time when the maximum degree of a graph is at most 2.

So our algorithm recursively solves the problem by finding a vertex v of maximum degree in the graph and if $d(v) \geq 3$ then recursively branching on two cases, in one case includes v in the independent set and none of its neighbors and in second case does not include v in the independent set and finally returns the maximum of the solutions returned by the two cases. In first case we remove $N[v]$ from the graph and in the second case we remove v form the graph. The detailed algorithm is described below.

Algorithm MaxIS(G, I**)**(*G is an undirected graph, I is an independent set of vertices*)
(Returns a maximum independent set of G containing all of the vertices of I. Initially the algorithm is called by MaxIS$(G, \emptyset)$.)

Step 0: If there is a vertex u of degree 0 or 1 in G then return $I_1 = MaxIS(G - N[u], I\{u\})$. Else go to Step 1.

Step 1: Find a vertex v of maximum degree in G. If $d(v) \leq 2$ then go to Step 4.

Step 2: Let I_1 and I_2 be the independent sets returned by the the following recursive calls:

1. $I_1 = MaxIS(G - \{v\}, I)$,
2. $I_2 = MaxIS(G - N[v], I \cup \{v\})$.

If $|I_1| \geq |I_2|$, return I_1 else return I_2.

Step 3: Find a maximum independent set I' for G and return $I' \cup I$.

The correctness of the algorithm is clear from the observations made above and the relation that

$$MIS(G) = \max\Big\{MIS(G - N[v]) \cup \{v\}, MIS(G - v)\}\Big\}.$$

The time complexity of the algorithm can be described by the following recurrences in n, where n is the number of vertices in the original graph.

$$T(n) \leq \max_{i \geq 3} \Big\{T(n-1) + T(n-(i+1)) + n^{O(1)}.$$

The worst recurrence occurs when $i = 3$ and this leads to the following theorem.

Theorem 2.9 MIS *can be solved in* $\mathcal{O}(1.3803^n n^{O(1)})$.

The analysis of this algorithm can be improved with an application of Measure and Conquer technique without making any changes to the algorithm. The idea of measure and conquer is to associate a *potential* and *measure* to the algorithm, and bound the number of nodes in the search tree generated by the algorithm as a function of this potential. This potential function can be any function in n or may be a function in some structural parameter of the graph. The intuition is that if we define a potential which is guided by a particular algorithm, then it will capture its performance more accurately. Hence a better upper bound on the number of nodes in the search tree can be obtained which will finally result in an improved time complexity of the algorithm. Finally, to obtain the running time of the algorithm in terms of n, we use an upper bound on the measure in terms of n and use this upper bound to convert the running time in terms of n. Here we illustrate this by analyzing MaxIS in terms of a measure which associates weights of value at most 1 to the vertices. We use different weights for vertices of different degree:

1. Weights to vertices of degree 0 is 0.

2. Weights to vertices of degree 2 is 0.5.
3. Weights to vertices of degree at least 3 is 1.

Let n_i denote the number of vertices of degree i in the graph G and $n_{\geq i}$ be the number of vertices of degree at least i in G. Hence the measure

$$\mu = \frac{n_2}{2} + n_{\geq 3} \leq n.$$

Now we write a generic recurrence when we branch on a vertex v of degree $d = d(v) \geq 3$. Notice that when we branch in the algorithm every vertex of the graph has degree at least 2. Let d_2, d_3 and $d_{\geq 4}$ be the number of neighbors of v of degree 2, 3 and at least 4 respectively (d_i or $d_{\geq i}$ could be equal to 0 too). This gives us the following recurrence in terms of μ on the number of nodes in the search tree. Here f represents the number of nodes in the recursion tree when the measure of the graph is μ.

$$f(\mu) \leq f\left(\mu - 1 - \frac{d_2 + d_3}{2}\right) + f\left(\mu - 1 - \frac{d_2}{2} - (d_3 + d_{\geq 4})\right).$$

The description of the recurrence is clear from the fact that when we delete a vertex then all its neighbors loses its degree by 1. The worst case recurrences occur when degree d is 3 or 4. For presentation we show all the recurrences possible when degree is 4 and 3. The corresponding recurrences are shown in Figure 2.6. The worst case recurrence of the set of equations in Figure 2.6 is $f(\mu) \leq f(\mu - 1) + f(\mu - 5)$ and it solves to 1.3248^{μ}. Now using the upper bound that $\mu \leq n$, we get the following theorem.

Theorem 2.10 MIS *can be solved in* $\mathcal{O}(1.3248^n n^{O(1)})$.

The running time of MIS can be improved further with some more reduction rules and much a more involved measure. We used different weights for vertices of different degree. *How do we decide on these weights ?* It is no magic, we write all possible recurrences symbolically and try to do "some kind of optimization". Generally, the recurrences for larger values of d are majorised by the ones for smaller values of d. That is, the solutions to the recurrences corresponding to larger values of d are upper bounded by the values to the solutions of the recurrences corresponding to smaller values of d. Hence we only consider all possible recurrences

$$f(\mu) \leq \max \begin{cases} f(\mu-3)+f(\mu-5) & \text{if } (4,0,4,0) \\ f(\mu-2.5)+f(\mu-5) & \text{if } (4,0,3,1) \\ f(\mu-2)+f(\mu-5) & \text{if } (4,0,2,2) \\ f(\mu-1.5)+f(\mu-5) & \text{if } (4,0,1,3) \\ f(\mu-1)+f(\mu-5) & \text{if } (4,0,0,4) \\ f(\mu-3)+f(\mu-4.5) & \text{if } (4,1,3,0) \\ f(\mu-2.5)+f(\mu-4.5) & \text{if } (4,1,2,1) \\ f(\mu-2)+f(\mu-4.5) & \text{if } (4,1,1,2) \\ f(\mu-1.5)+f(\mu-4.5) & \text{if } (4,1,0,3) \\ f(\mu-3)+f(\mu-4) & \text{if } (4,2,2,0) \\ f(\mu-2.5)+f(\mu-4) & \text{if } (4,2,1,1) \\ f(\mu-2)+f(\mu-4) & \text{if } (4,2,0,2) \\ f(\mu-3)+f(\mu-3.5) & \text{if } (4,3,1,0) \\ f(\mu-2.5)+f(\mu-3.5) & \text{if } (4,3,0,1) \\ f(\mu-3)+f(\mu-3) & \text{if } (4,4,0,0) \\ f(\mu-2.5)+f(\mu-4) & \text{if } (3,0,3,0) \\ f(\mu-2.5)+f(\mu-3.5) & \text{if } (3,1,2,0) \\ f(\mu-2.5)+f(\mu-3) & \text{if } (3,2,1,0) \\ f(\mu-2.5)+f(\mu-2.5) & \text{if } (3,3,0,0) \end{cases}$$

Figure 2.1: Recurrence when degree of branching vertex is 3 and 4. Here tuple in the bracket represents $(d, d_2, d_3, d_{\geq 4})$

for smaller values of d, which in turn bounds the number of recurrences we need to consider. Then we either use the quasi-convex programming of Eppstein [90] or random local search and find values for the weights which (almost) minimizes the value of μ, among those which satisfy all the recurrences. This can be done using standard programming in any modern mathematical softwares (say Matlab, Maple or Mathematica).

Part II

Fixed Parameter Tractable Algorithms and Intractability

3
Undirected Feedback Vertex Set

The Feedback Vertex Set (FVS) problem is a vertex deletion problem where the objective is to make the graph acyclic by deleting as few vertices as possible. More formally the problem is defined as follows:

> Feedback Vertex Set (FVS): Given an undirected graph $G = (V, E)$, and an integer parameter k, is there a subset of at most k vertices whose removal results in an acyclic graph?

Feedback Vertex Set is one of the well known NP-complete problems in undirected and directed graphs. But the edge counterpart of Feedback Vertex Set problem known as Feedback Edge Set problem, where the objective is to delete as few edges as possible to make the graph acyclic is polynomial time solvable in undirected graphs. This follows from its equivalence to the Minimum Spanning Tree problem.

The parameterized version of the Feedback Vertex Set problem can be shown to be fixed parameter tractable using the graph minor theorem. But the algorithm based on graph minors is highly nonconstructive. This has led to the development of many constructive algorithms beginning with an $\mathcal{O}((2k+1)^k n^2)$ algorithm [79]. If we are willing to accept uncertainty about the correctness of the answer, one can solve this problem in $O(4^k kn)$ time by a randomized algorithm presented in [21]. In this chapter we reduce the function $f(k)$ for the Feedback Vertex Set problem by developing an algorithm whose running time is $O\left(\left(\frac{12\log k}{\log\log k}+6\right)^k n^\omega\right)$ where ω is the exponent in the runtime for Matrix Multiplication. In the rest of the chapter we use *fvs* to denote a feedback vertex set of a given graph and FVS to denote the problem FEEDBACK VERTEX SET.

Our results are based on a structural characterization of feedback vertex set in graphs with minimum degree 3. It is well known that an undirected graph possesses a cycle of logarithmic length if its minimum or average degree is at least 3. In this chapter, we obtain a similar result on the existence of short cycles in graphs having a small feedback vertex set. We show that if an undirected graph on n vertices with minimum degree at least 3 has a fvs on at most $\frac{1}{3}n^{1-\epsilon}$ vertices, then there is a cycle of length at most $\frac{6}{\epsilon}$ (for $\epsilon \geq 1/2$, we can even improve this to just 6). This is one of the main contributions of this chapter and is of independent interest. Using this structural result, we obtain a faster algorithm for solving the FVS problem.

Most of the known algorithms for the problem use the bounded search tree technique. In particular, they work by finding a short cycle in the graph after some preprocessing, and branching recursively on each vertex of the short cycle. The algorithm presented here also adheres to this paradigm. The correctness and efficiency of our algorithm is based on the new graph theoretical result (mentioned before) which connects minimum fvs size and the length of the shortest cycle .

We also show that the weighted feedback vertex set problem is solvable in essentially the same time if each vertex has real weight at least 1. In the general case when the weights are arbitrary real numbers, we show that the problem is unlikely to be fixed parameter tractable.

Throughout the chapter, we use $\delta(G)$ to denote the minimum degree of a graph G and $g(G)$ to denote its girth, i.e., the length of a shortest cycle in the graph.

Organization of the rest of the Chapter: Given G and k, one can construct a graph G', in polynomial time, with $\delta(G') \geq 3$ such that G has a fvs of size at most k if and only if G' has a fvs of size at most k. Section 3.1 describes this preprocessing step. It also describes a generic, short cycle based branching algorithm which will be used as a template by the rest of the algorithms we develop in this chapter.

Section 3.2 proves one of the main structural results relating the size of fvs and short cycles. It also derives a simple FPT algorithm for FVS problem as a consequence. Section 3.3 presents the proof of a generalization of structural result presented in the Section 3.2 and gives a description of a faster algorithm for FVS based on it.

Section 3.4 investigates the parameterized complexity of weighted feedback vertex set. Section 3.5 describes improved algorithms for some special classes of graphs like bounded degree graphs, regular graphs and almost regular graphs.

Section 3.6 introduces a new algorithm for the FVS problem. This is based on the degree sequence of a graph. Also, in this section, we present an $O((8\log k + 2)^k n^\omega)$ algorithm based on a result relating girth and the number of vertex disjoint cycles.

Finally Section 3.7 concludes with some remarks and future directions.

3.1 Preliminaries

In this section, we first describe some preprocessing rules which removes non-essential vertices from the input graph without affecting the size of a minimum fvs. Then, we present a generic algorithm for finding a fvs which is going to be the template of our main algorithmic results.

3.1.1 Preprocessing

The following is well known in the literature on FVS problems. See, for example, [20] for proofs.

Lemma 3.1 *Let G be an undirected multi-graph. Perform the following steps as long as possible.*

1. *If G has a vertex of degree ≤ 1, remove it (along with the incident edge if any).*
2. *If G has a vertex x of degree 2 adjacent to vertices y and z, $y \neq x$ and $z \neq x$, short circuit by removing x and joining y and z by a new edge (even if y and z were adjacent earlier).*

Let G' be the resulting multi-graph. Then G has a feedback vertex set of size at most k if and only if G' has a feedback vertex set of size at most k.

Clearly the graph G' is such that each component of G' has minimum degree at least three unless that component is *either* an empty graph *or* a graph on one vertex with a self loop (in which case that component has a feedback vertex set of size 1). G' can be constructed in $O(m)$ steps where m is the number of edges in G.

Lemma 3.2 *[20] Given an undirected multi graph $G = (V, E)$ on m edges, in $O(m)$ time we can produce a multi-graph G' with minimum degree 3 such that G has a fvs of size k if and only if G' has a fvs of size k.*

We also note that

Lemma 3.3 *Given an undirected multi graph $G = (V, E)$ on n vertices with $\delta(G) \geq 3$, removing a vertex v from G can be achieved in $O(n)$ time.*

We recall the following well-known algorithmic results.

Lemma 3.4 *[64] Given an undirected multi graph G, we can test whether G has a cycle or not in $O(n)$ time, where n is the number of vertices in G.*

Lemma 3.5 *[149] Given an undirected multigraph G, a shortest cycle (if there is any) in G can be found in $O(\min\{mn, n^\omega\})$ time where n^ω is the running time of the best-known algorithm for multiplying two n by n matrices.*

Lemma 3.6 *[149] Given an undirected graph G on n vertices, a cycle of length at most $g(G) + 1$ in G can be found in $O(n^2)$ time.*

3.1.2 A Generic Algorithm

The following generic algorithm forms the basis of our main algorithmic results. Here, G is an undirected multigraph and $k \geq 0$. The algorithm returns YES and a feedback vertex set of size at most k in G if there is one and returns NO otherwise.

Algorithm GFBVS(G, k)

- ***Step*** $0'$***:*** If G is acyclic, then answer YES and return $\emptyset$.
- ***Step*** 0***:*** If $k = 0$ and G contains a cycle, then answer NO and EXIT.
- ***Step*** 1***:*** Apply Lemma 3.1 to get G'.

- ***Step*** 2***:*** Find a shortest cycle C in G'. (C could possibly be of length 1 or 2.)
- ***Step*** 3***:*** If for some vertex $v \in C$, GFBVS$(G' - v, k-1)$ is true then answer YES and return $\{v\} \cup$ GFBVS$(G' - v, k-1)$, else answer NO.

The correctness of the algorithm follows from Lemma 3.1 and the fact that any feedback vertex set must contain a vertex from every cycle in the graph.

Furthermore, if $g(G') \leq g$ for all the graphs G' used in Step 2 of the recursive calls, then the overall algorithm takes $O(g^k n^\omega)$ time. This is because the recursion tree at Step 3 has a branching factor of at most g and depth at most k, and Step 2 takes $O(n^\omega)$ time from Lemma 3.5. Steps 0 and 1 each take $O(m)$ time by Lemma 3.4. Also, in Step 2, instead of finding a shortest cycle, if we find a cycle of length at most $g+1$, then by Lemma 3.6, the running time of the algorithm will be $O((g+1)^k n^2)$ time. Thus we have

Lemma 3.7 *Let G be an undirected graph, and let g be the maximum size of the girth of the graphs G' used in Step 2 of* **GFBVS**(G, k)*. Then we can find a feedback vertex set of size at most k in G (or determine its absence) in $O(g^k n^\omega)$ time or in $O((g+1)^k n^2)$ time.*

Erdös and Posa [92] observed that girth of any undirected graph G with minimum degree at least 3 is bounded by $2\log n$. Given such a graph, one can find in $O(n)$ time a cycle of length at most $2\log n$ by growing a Breadth First Search (BFS) tree till the first non-tree edge is encountered. Thus, we get

Lemma 3.8 *Any graph G with minimum degree at least* 3 *has a cycle of length at most* $2\log n$ *and such a cycle can be found in $O(n)$ time where n is the number of vertices in the graph G.*

So, in Step 2 of the generic algorithm if we find just a cycle of length at most $2\log n$ (which may not necessarily be the shortest cycle), then from Lemma 3.7 and Lemma 3.8, we have

Theorem 3.1 *Given a graph G on n vertices, and an integer parameter k, we can determine whether or not G has a feedback vertex set of size at most k in $O((2\lg n)^k n + m)$ time, or in $O((4k\log k)^k n + nm)$ time.*

The second bound follows from the observation that

$$(2\log n)^k \leq (4k\log k)^k + n$$

for all n and $k \leq n$.

3.2 Feedback Vertex Set and Girth - I

We first present the proof of the following theorem and discuss the tightness of our result.

Theorem 3.2 *Let G be a graph on n vertices with minimum degree at least 3, having a feedback vertex set of size k, such that $(n-k) > 4 \cdot \binom{k}{2}$. Then $g(G) \leq 6$.*

Then we derive a simple FPT algorithm as its consequence. The arguments in Theorem 3.2 are based on the following lemma. We present two different proofs of the lemma as they show different facets of degree sequences in a forest.

Lemma 3.9 *Let $T = (V, E)$ be a forest on N vertices. Let $M' = \{(i,j) \in E | \; deg_T(i) = deg_T(j) = 2\}$ and $L = \{a \in V \mid deg_T(a) \leq 1\}$. Then there exist $M \subseteq M'$, such that M is a matching and $|W = L \cup M| \geq N/4$. Here $deg_T(x)$ represents the degree of x in T.*

Proof: Without loss of generality assume that T is a tree, otherwise we can apply the result on each tree of the forest and combine them to get the desired result. Now let v be a vertex of degree not equal to 2. Root the tree at v and direct the edges away from the root, call it T_v. Let D_2 be the set of degree 2 (undirected degree) vertices of T and $D_{\geq 3}$ denotes the set of vertices of degree at least 3 in T. Every connected component of the induced graph $T_v[D_2]$ is either an isolated vertex or a directed path. Let M be a matching on degree 2 vertices consisting of all the alternate edges starting from the vertices of in-degree 0 of all the paths of $T_v[D_2]$ ignoring their direction. Clearly $M \subseteq M'$. Define S to be the set of vertices in D_2 unmatched by any edge in M. Note that each vertex in S is either an isolated vertex of $T_v[D_2]$ or the last vertex in an even length directed path of $T_v[D_2]$. See Figure 3.1. Now with each vertex $u \in S$, associate its unique child in T_v, whose degree is either 1 or at least 3 in T. Note that this association is

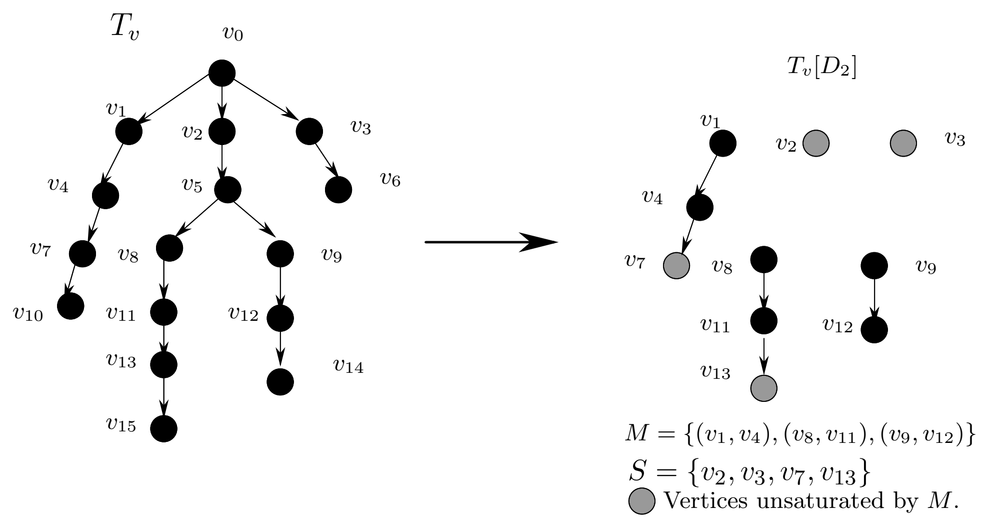

Figure 3.1: Illustration of Lemma 3.9

injective, as in a rooted tree we have a unique parent for every vertex other than root. This implies that $|S| \leq |L| + |D_{\geq 3}|$. It is well known that the number of vertices of degree at least 3 in a tree is smaller than the number of leaves of the tree. So, we have $|D_{\geq 3}| < |L|$ and $|S| \leq |L| + |D_{\geq 3}| \leq 2|L|$. This gives

$$N = |L| + 2|M| + |S| + |D_{\geq 3}| < |L| + 2|M| + 2|L| + |L| \leq 4|L| + 2|M|.$$

Dividing both sides by 4 gives $N/4 < |L| + |M|/2 \leq |W = L \cup M|$. This completes the proof. $\square$

Proof: (**Alternate proof of Lemma 3.9.**) Let d_0, d_1 and d_2 denote, respectively, the number of vertices in T of degree 0, 1 and 2 and let d_g be the number of vertices in T of degree at least 3. Let $d_{01} = d_0 + d_1$. Then the following claims are easy to see.

Claim 1: $d_{01} + d_2/2 \geq N/2 + 1$.

Claim 2: $d_g \leq d_{01} - 2$

Claim 3: Let R be the set of degree 2 vertices of T having only high degree (3 or more) neighbors. Then $|R| \leq d_g - 1$.

The first claim follows from the fact that

$$\begin{aligned} d_0 + d_1 + 2d_2 + 3(N - d_0 - d_1 - d_2) &\leq 2 \mid E \mid + d_0 \\ &\leq 2(N - d_0 - 1) + 2d_0 \\ &\leq 2N - 2. \end{aligned}$$

Hence $N + 2 \leq 2d_0 + 2d_1 + d_2$ from which the claim follows.

The second claim follows from the fact that $N = d_g + d_{01} + d_2 = d_g + (2d_{01} + d_2) - d_{01} \geq d_g + N + 2 - d_{01}$ (by Claim 1).

To prove the third claim, we consider the graph induced by vertices in R and its (high degree) neighbors. The graph has at least $2|R|$ edges as every vertex in R has two high degree neighbors and no pair of vertices in R is adjacent. It has at most $|R| + d_g - 1$ edges as it is a forest on at most $|R| + d_g$ vertices. So we have, $2|R| \leq |R| + d_g - 1$ proving Claim 3.

Assume that $d_{01} < N/4$, otherwise the lemma follows.

By Claim 1, $d_2 \geq N - 2d_{01} + 2$. Also, by Claims 2 and 3, of the at least $N - 2d_{01} + 2$ degree 2 vertices, at most $d_{01} - 3$ of them have no low degree neighbors. That is, at least $N - 3d_{01} + 5$ of them have some low degree neighbors. As a result, there are at least $N - 4d_{01} + 5$ degree 2 vertices each of which has some degree 2 neighbor in T. From this, we can get a matching of at least $N/2 - 2d_{01} + 2$ edges joining degree 2 vertices. Hence, $|M| \geq N/2 - 2d_{01} + 2$. Hence, we have

$$d_{01} + |M| \geq N/2 - d_{01} + 2 \geq N/4$$

since $d_{01} < N/4$. Since, the set of edges M used is a matching of edges joining degree 2 vertices, this proves the lemma. □

Proof: (**of Theorem 3.2**) If G has a loop or a multiple edge, then $g(G) \leq 2$. Hence, we assume that G is a simple graph having no self-loops and parallel edges. Let F be a fvs of size k in G and let T denote the induced forest $G[V - F]$ on $N = n - k$ vertices. Apply Lemma 3.9 to T to get W (mentioned in the Lemma 3.9). Now with every element $a \in W$, we associate a (unordered) pair of vertices of F as follows:

Case 1: $a \in L$, i.e., a is a vertex of degree 0 or 1.
Since the degree of a is at least 3 in G, a has at least two neighbors in F. We pick arbitrarily two of these neighbors and associate them with a. We use $\{x_a, y_a\}$ to denote the pair associated with vertex a.

Case 2: $a = (u, v)$ is an edge from M.
Since each of u and v has degree at least 3 in G, each of them has at least one neighbor in F. We pick one neighbor (in F) of each u and v and associate them with a. We use $\{x_u, x_v\}$ to denote the pair associated with $a = (u, v)$. Note that possibly $x_u = x_v$.

Suppose there is a pair $\{x, y\}$ associated with some $a \in W$ such that $x = y$. In that case, a should be an edge $(u, v) \in M$. Thus, we get a 3-cycle (x, u, v, x) in G proving that $g(G) \leq 6$. Hence, we assume that every selected pair $\{x, y\}$ is such that $x \neq y$. Since the number of such pairs is at most $\binom{k}{2}$ and as $|W| \geq \frac{N}{4} > \binom{k}{2}$, the association map is not injective. That is, there are $a_1, a_2 \in W$, $a_1 \neq a_2$ such that both a_1 and a_2 are associated with some pair $\{x, y\}$ with $x \neq y$. The following cases arise.

- Both a_1 and a_2 are vertices of degree at most 1. In that case, we get a 4-cycle (x, a_1, y, a_2, x), thereby proving that $g(G) \leq 4$.

- Both a_1 and a_2 are edges (u_1, v_1) and (u_2, v_2) from M such that x is a neighbor of u_1 and u_2 and y is a neighbor of v_1 and v_2. In this case, we get a cycle $(u_1, x, u_2, v_2, y, v_1, u_1)$ of length 6. This leads to a cycle of length 6, thereby proving that $g(G) \leq 6$.

- a_1 is a vertex of degree ≤ 1 and a_2 is an edge (u, v) from M. Also, x is a neighbor of a_1 and u and y is a neighbor of a_1 and v. This gives rise to a cycle (a_1, x, u, v, y, a_1) of length 5, proving again that $g(G) \leq 6$.

In all cases we have that $g(G) \leq 6$ proving Theorem 3.2. □

Corollary 3.1 *If a graph on n vertices with minimum degree 3 has a feedback vertex set of size at most $\sqrt{n/2}$, then $g(G) \leq 6$.*

Remark 1: Corollary 3.1 is tight in the sense that there are graphs G satisfying the hypothesis of the corollary with $g(G) = 6$, as the following example (Figure 3.2) shows.

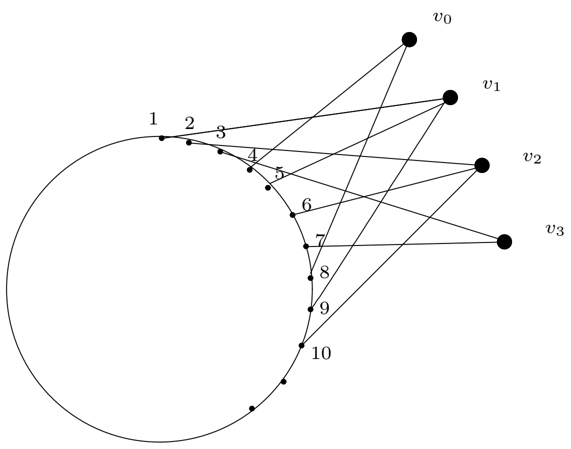

Figure 3.2: Graph G with $g(G) = 6$, fvs of size at most 5 and $\delta(G) \geq 3$.

Let $G = (V, E)$ be a graph on $n \geq 63$ vertices such that n is a multiple of 4. The graph has a cycle C of length $n - 4$ on vertices 1 to $n - 4$. The remaining 4 vertices are named v_0, v_1, v_2 and v_3. The vertex v_i is adjacent to all vertices j in the cycle such that $j\ mod(4) = i$.

It is easy to see that $g(G) = 6$, and $|F| \leq 5$ as $\{1, v_0, v_1, v_2, v_3\}$ is a feedback vertex set of G.

Corollary 3.2 *Let G be a graph on n vertices with minimum degree 3 having a feedback vertex set of size at most k. Then, $g(G) \leq \max\{6, 4\lg k + 2\}$.*

Proof: If $k \leq \sqrt{n/2}$, then $g \leq 6$ by the previous corollary. Otherwise, $n \leq 2k^2$ and hence, by Lemma 3.8, we have $g \leq 2 \log n \leq 4 \log k + 2$. □

Based on the Corollary 3.2 we derive the next theorem.

Theorem 3.3 *Let G be an undirected graph on n vertices. Then, we can determine whether or not G has a feedback vertex set of size at most k (and find one if there is) in $O\left(\max\{6, 4\log k + 2\}^k n^\omega\right)$ time.*

Proof: Modify the Step 2 in our generic algorithm as follows:

- ***Modified Step 2:*** Find a shortest cycle C in G'. If $k \leq \sqrt{n/2}$ and $g > 6$, then answer NO.

Now use Corollary 3.2 and apply Lemma 3.7 over **GFBVS**(G, k) with modified step 2 to get Theorem 3.3. □

3.3 Feedback Vertex Set and Girth - II

We can generalize Theorem 3.2 and Corollary 3.1 to prove upper bounds for girth in graphs having larger feedback vertex sets than assumed in Theorem 3.2. First we will need the following result of Alon, Hoory and Linial [9].

Theorem 3.4 *[9] Any graph $G = (V, E)$ on n vertices with average degree d, contains a cycle of length $\leq 2\log_{d-1} n + 2$.*

Using this result, we prove our generalized theorem.

Theorem 3.5 *Let $0 < \epsilon < 1$. Let G be a graph on n vertices such that (a) $\delta(G) \geq 3$, (b) $n \geq \lceil 3^{\frac{1}{\epsilon}} \rceil$, and (c) G has a fvs of size at most $\frac{1}{3}\, n^{1-\epsilon}$. Then G has a cycle of length at most $6/\epsilon$, that is, $g(G) \leq 6/\epsilon$.*

Proof: We can assume that $\epsilon < 1/2$ as otherwise the theorem follows from Corollary 3.1. Let G be a graph on $n \geq n_0 = \lceil 3^{1/\epsilon} \rceil$ vertices with minimum degree 3 and having a feedback vertex set F of size $k \leq \frac{1}{3} n^{1-\epsilon}$. As before, let T denote the induced forest on the remaining $N = n - k$ vertices in G.

We construct a new multi-graph G' with $V(G') = F$ as follows. The edges of G' are included as follows. For every $a \in W$ (where W is the set obtained by applying Lemma 3.9 on $T = G[V - F]$), we include an edge between x_a and y_a ($x_a = y_a$ possibly) where $\{x_a, y_a\}$ is the the pair associated with a in F in the proof of Theorem 3.2. By Lemma 3.9, we know that W is of size at least $N/4$. It follows that G' has at least $N/4$ edges and hence its average degree is at least $N/2k$ as $|V(G')| = |F| = k$.

Note that if G' has a cycle of length at most g, then G has a cycle of length at most $3g$, as any edge of the cycle in G' can be replaced by a path of length at most 3 in the original graph G.

By Theorem 3.4, G' has a cycle of length at most

$$(2\log_{(N/2k)-1} k) + 2 = \frac{2\log k}{\log((N/2k)-1)} + 2.$$

This implies that G has a cycle of length at most

$$\frac{6\log k}{\log((N/2k)-1)} + 6.$$

After substituting $N = n - k$ and $k \le \frac{1}{3}n^{1-\epsilon}$, we get

$$\begin{array}{rcll} g(G) & \le & \dfrac{6\log(n^{1-\epsilon}/3)}{\log(\frac{3}{2}n^{\epsilon}-\frac{3}{2})} + 6 & \\ & < & \dfrac{6(1-\epsilon)\log n}{\log n^{\epsilon}} + 6 & \quad \text{for } n \ge n_0. \\ & \le & \dfrac{6(1-\epsilon)}{\epsilon} + 6 = \dfrac{6}{\epsilon} & \end{array}$$

which is what we wanted to show. □

Corollary 3.3 *Let G be a graph on n vertices with minimum degree ≥ 3 having a feedback vertex set of size at most k. Then, for every ϵ $(0 < \epsilon < 1)$ such that $n \ge \lceil 3^{1/\epsilon} \rceil$, $g(G) \le \max\{\frac{6}{\epsilon}, \frac{2\log 3k}{1-\epsilon}\}$.*

Proof: If $k \le n^{1-\epsilon}/3$, then $g \le 6/\epsilon$ by the previous theorem. Otherwise, $n \le (3k)^{\frac{1}{1-\epsilon}}$ and hence, by Lemma 3.8, $g \le \frac{2\log 3k}{1-\epsilon}$. □

For fixed values of ϵ, the lower bound on n (required to apply Corollary 3.3) is also fixed. Hence, by applying Lemma 3.7 to **GFBVS**(G, k), with the following modified step 2,

- ***Modified Step 2:*** Find a shortest cycle C in G' of lnegth g. If $k \le \frac{1}{3}n^{1-\epsilon}$ and $g > \frac{6}{\epsilon}$, then answer NO.

we get the following theorem.

Theorem 3.6 *Let G be an undirected graph on n vertices. Then, for every fixed ϵ, $0 < \epsilon < 1$, we can determine (and find one, if exists) whether or not G has a feedback vertex set of size at most k in $O\left(\max\left(\frac{6}{\epsilon}, \frac{2\log 3k}{1-\epsilon}\right)^k n^{\omega}\right)$ time.*

3.3.1 A Faster Algorithm

In this Section we give a faster FPT algorithm for FVS problem. Our improved algorithm is based on the fact that Theorem 3.5 gives us a kernel of size $(3k)^{\frac{1}{1-\epsilon}}$ for every ϵ, $0 < \epsilon < 1$, in time $O(\left(\frac{6}{\epsilon}\right)^k n^{\omega})$. The new algorithm makes use of this reduction by choosing a proper ϵ and obtains a kernel. After that, it works by enumerating all subsets of size at most k instead of branching on a short cycle. The algorithm is presented below. As usual, G is an undirected multigraph and $k \geq 0$ is an integer.

Algorithm Mod-FBVS(G, k)

- ***Step 0 :*** If G is acyclic answer YES or if $k = 0$ answer NO.
- ***Step 1 :*** Apply Lemma 3.1 to get G'.
- ***Step 2:*** Find a shortest cycle C in G'. Let g be its length.
- ***Step 3a:*** If $g < 6\left(\frac{\log k+\log\sqrt{\log k}}{\log\sqrt{\log k}}\right)$, then if for some $v \in C$, Mod-FBVS($G' - v, k-1$) is true then answer YES and return $\{v\}\cup$ Mod-FBVS($G' - v, k-1$), else answer NO.
- ***Step 3b:*** If $n > 9k\sqrt{\log k}$ then return NO. Else try all possible k-subsets of $V(G)$ as a possible feedback vertex set of G and say YES, if any such subset is a fvs and return that subset, else say NO.

Correctness of the algorithm Mod-FBVS follows from its description. When we reach Step 3b of the algorithm, we have $g \geq 6\left(\frac{\log k+\log\sqrt{\log k}}{\log\sqrt{\log k}}\right)$. Then, we use Theorem 3.5 by choosing $\epsilon = \frac{\log\sqrt{\log k}}{\log k+\log\sqrt{\log k}}$. Note that $n \geq \lceil 3^{1/\epsilon}\rceil$ and by Lemma 3.8,

$$\begin{aligned} 2\log n &\geq g \geq 6\left(\frac{\log k+\log\sqrt{\log k}}{\log\sqrt{\log k}}\right) \\ \log n &\geq \frac{3}{\epsilon} \\ n &\geq 8^{\frac{1}{\epsilon}} \geq \lceil 3^{\frac{1}{\epsilon}}\rceil. \end{aligned}$$

Thus, by Theorem 3.5 we have

$$\frac{1}{3}n^{1-\epsilon} < k \Longleftrightarrow n < (3k)^{\frac{1}{1-\epsilon}} \Longleftrightarrow n < (3k)^{\frac{\log k + \log\sqrt{\log k}}{\log k}} \leq 9k\sqrt{\log k},$$

or there is no fvs of size at most k. So either the girth is bounded by $\frac{12\log k}{\log\log k} + 6$ or we have a kernel of size $\leq 9k\sqrt{\log k}$. So the time complexity of the algorithm is bounded by:

$$\max\left\{ \left(\frac{12\log k}{\log\log k} + 6\right)^k n^\omega, \binom{9k\sqrt{\log k}}{k} n^2 \sim (9e\sqrt{\log k})^k n^2 \right\}.$$

Since the first function is asymptotically larger, we use it to bound the time complexity. Combining all these, we get the proof of the following theorem.

Theorem 3.7 *Let G be an undirected multi-graph on n vertices. Then, we can determine whether or not G has a fvs of size at most k in time*

$$O\left(\left(\frac{12\log k}{\log\log k} + 6\right)^k n^\omega\right).$$

3.4 Weighted Feedback Vertex Set

The Weighted Feedback Vertex Set problem ($WFVS$ for short) is: given an undirected graph $G = (V, E)$, a weight function $w : V \to \mathbf{R}^+$, and $k \in \mathbf{R}^+$, find a feedback vertex set F with total weight at most k. The weight of F is defined as the sum of weights of $v \in F$.

In the weighted case, the preprocessing described in Lemma 3.1 cannot be applied as such because it is possible that every minimum weight fvs contains some degree two vertex. However, if we assume that $w(v) \geq 1$ for every v, then we can modify the preprocessing as follows. Given a graph G with a vertex weight function w, we repeatedly remove vertices of degree 1 to transform G into G'' with minimum degree ≥ 2. Then, for every path P in G'' joining two vertices x and y of larger (≥ 3) degrees such that each internal vertex of P has degree two, we replace P by the path xzy where z is an internal vertex of P having minimum weight among all internal vertices of P. Let G' be the resulting weighted graph. The weights of vertices surviving in G' are the same, as assigned to them in G.

Now it is easy to verify that G has a feedback vertex set of weight at most k if and only if G' has a fvs of weight at most k. Let us call such a graph having minimum degree 2 with each degree 2 vertex connected to two vertices of larger degree as a *branchy graph.* One can easily adapt Theorem 3.5 for branchy graphs by using the fact that every vertex of degree 2 has both its neighbors of degree at least three and obtain its weighted version that shows that if a weighted branchy graph G (with weight function w such that $w(v) \geq 1$) has a fvs of weight at most $\frac{1}{3}n^{1-\epsilon}$, where $0 < \epsilon < 1$, then $g(G) < \frac{12}{\epsilon}$, provided $n \geq \lceil 3^{\frac{1}{\epsilon}} \rceil$.

For the weighted case, we modify the algorithm Mod-FBVS by reducing G to a branchy graph (as described before) in Step 1. We then look for a cycle of length

$$g < 12 \left(\frac{\log k + \log \sqrt{\log k}}{\log \sqrt{\log k}} \right)$$

in Step 3a of the algorithm. As before, the algorithm either finds a short cycle and branches on the vertices of the cycle or enumerates all subsets of size at most k. In the first case, since each vertex picked for branching has weight at least 1, the depth of the recursion is at most k. Also, we can show that in Step 3b, $n = O(k\sqrt{\log k})$, as in the unweighted case by using the fact that every vertex has weight at least 1. Thus, we have an analogue of Theorem 3.7

Theorem 3.8 *Given an undirected graph $G = (V, E)$, a positive real parameter k and a weight function w from V to $\mathbf{R}^+$ such that for every $v \in V$, $w(v) \geq 1$, we can determine whether or not G has a feedback vertex set of weight at most k in time*

$$O\left(\left(\frac{24 \log k}{\log \log k} + 12 \right)^k n^\omega \right).$$

General-WFVS is the problem of finding a fvs of weight at most k, when the weights of the vertices are arbitrary real numbers. We show that the problem is not fixed parameter tractable unless $P = NP$ by proving that it is NP-complete for any fixed $k > 0$. We can give a direct reduction from the NP-complete, unweighted FVS problem on undirected graphs to General-WFVS with $k = 1$, by defining the weight function w to be $w(v) = 1/k$ for all $v \in V$. In fact this implies that there cannot be a $f(k)n^{O(1)}$ or even $n^{O(k)}$ time algorithm for General-WFVS problem unless $P = NP$.

Theorem 3.9 *General-WFVS problem is not fixed parameter tractable unless $P = NP$.*

3.5 Algorithms for FVS in Special Graph Classes

Here we show that the dependence on the parameter k can be made a simple exponential function for bounded degree graphs and regular graphs. Note that the FVS problem remains NP complete even for planar graphs and bounded degree graphs [127, 159, 229]. For planar graphs, FPT algorithms with time complexity sub-exponential in the parameter k are known [164].

This improvement is based on the observation that for such classes of graphs, any fvs should have size linear in n. It is similar to the lower bound of $|F| \geq (n+2)/(\Delta+1)$ obtained by Voss [220] on the size of any fvs F of any G with $\delta(G) \geq 3$. For bounded Δ, this is linear in n.

Lemma 3.10 *Let $G(V, E)$ be a graph on n vertices with minimum degree $\delta \geq 3$ and maximum degree Δ. Then the size of the minimum feedback vertex set for G is greater than $n(\delta-2)/2(\Delta-1)$.*

Proof: Let F be a minimum feedback vertex set for G, and let E_F be the set of edges with at least one end point in F. Since $G - F$ is a forest, there are at most $n - |F| - 1$ edges in $G - F$. Thus

$$\Delta|F| \geq |E_F| \geq |E| - n + |F| + 1 > n\delta/2 - n + |F|$$

which implies

$$(\Delta - 1)|F| > n(\delta - 2)/2$$

or $|F| > n(\delta-2)/2(\Delta-1)$. □

Using this lemma, we give improved algorithms for regular, almost regular (defined later) and bounded degree graphs.

3.5.1 Regular Graphs

Setting $\delta = \Delta = r \geq 3$ in Lemma 3.10, we get

Corollary 3.4 *Let $G(V, E)$ be an r-regular graph on n vertices with $r \geq 3$. Then the size of the minimum feedback vertex set for G is more than $n/4$.*

Using Corollary 3.4, we can obtain the following FPT algorithm for regular graphs, based on the technique of reduction to a kernel.

> If G is 1-regular, then G is acyclic. If G is 2-regular, then G is a vertex disjoint collection of cycles. If the number of cycles is at most k, then answer YES and output a solution obtained by picking one (arbitrary) vertex from each cycle and answer NO otherwise.
>
> Otherwise, G is r-regular for some $r \geq 3$. If $k \leq n/4$, then answer NO, otherwise $n < 4k$ and try all k element subsets S of V to check whether $G - S$ is acyclic. If the answer is yes for any subset S then answer YES (S is a feedback vertex set of size k) and answer NO otherwise.

The correctness follows from the corollary above. Also the running time is dominated by $\binom{4k}{k} kn = O((4e)^k k^2)$. Thus we have

Theorem 3.10 *There is an $O((4e)^k k^2 + n)$ time algorithm to determine whether a given r-regular graph on n vertices has a fvs of size at most k.*

3.5.2 Almost Regular Graphs

Let us call a graph G *almost regular* if $\Delta(G) - \delta(G) \leq c$ for some constant c. Then we have, from Lemma 3.10

Corollary 3.5 *Let $G(V, E)$ be an almost regular graph on n vertices where $3 \leq \delta$ and $\Delta - \delta \leq c$. Then the size of the minimum feedback vertex set for G is more than $n(\Delta - c - 2)/2(\Delta - 1)$.*

Applying this lower bound and making use of the preprocessing described in Section 3.1, we get the following

Theorem 3.11 *For every constant $c > 0$, there exists a constant $d = d(c) > 1$ (depending on c) and a $O(d^k kn)$ time algorithm to determine whether an almost regular G (with $\delta \geq \Delta - c$) has a fvs size at most k.*

3.5.3 Bounded Degree Graphs

Here, we assume that G is a graph for which Δ is bounded above by an arbitrary but fixed positive integer c. This is a special case of almost-regular graphs. Hence, as a corollary of Theorem 3.11, we obtain the following :

Theorem 3.12 *For each positive integer c, there is some positive constant $d = d(c) > 1$ (depending on c) such that there is an $O(d^k kn)$ time algorithm to determine if a given G on n vertices with $\Delta(G) \leq c$ has a fvs of size at most k.*

3.6 Some Simple FPT Algorithms

In this section we make two additional remarks on finding a fvs of size at most k in an undirected graph. The first one introduces a new approach to this problem based on a simple observation connecting the degree sequence of a graph and feedback vertex sets. The second one follows from a result connecting girth and the number of vertex disjoint cycles.

3.6.1 Degree Sequence Algorithm

Given an undirected graph G and an integer $k \geq 0)$, the following algorithm returns a fvs of size at most k in G if there is one and returns NO otherwise.

Algorithm Deg-FBVS(G, k)

0. If G is acyclic, then answer YES and return $\emptyset$.

1. If $k = 0$ and G contains a cycle, then answer NO and EXIT.

2. If $n \leq 3k$ then try all possible k-subsets of veretx set V as a possible feedback vertex set of G and say YES, if any such subset is a fvs and return that subset, else say NO.

3. $G' \leftarrow$ Preprocess(G) using Lemma 3.2.

4. Sort the vertices in decreasing order $\{v_1, v_2, \cdots, v_n\}$ of their degrees.

5. If for some v_i, $1 \leq i \leq 3k$, Deg-FBVS$(G' - v_i, k-1)$ is true then answer YES and return $\{v\} \cup$ Deg-FBVS$(G' - v_i, k-1)$, else answer NO.

Let $\{v_1, v_2, \ldots, v_n\}$ be a decreasingly sorted order of the vertices (mentioned in Step 4 of the algorithm). Let V_h denote the set $\{v_1, \ldots, v_{3k}\}$. The correctness of the algorithm follows from the following lemma.

Lemma 3.11 *Any fvs F of size at most k must have $F \cap V_h \neq \emptyset$.*

Proof: To prove this lemma we need the following simple claim proved in [188]. We give the proof for the sake of completeness.

Claim 3.1 *Let $G = (V, E)$ be a graph on n vertex with minimum degree 3, then for every feedback vertex set (F) of G, we have*

$$\sum_{v \in F}(d(v) - 1) \geq |E| - |V| + 1.$$

Proof: The proof follows from the following inequality:

$$\sum_{v \in F} d(v) + |V| - |F| - 1 \geq |E|$$

□

Now we proceed to prove the lemma using Claim 3.1. Suppose F is a fvs of size k with $F \cap V_h = \emptyset$. It then follows that

$$\sum_{i=1}^{3k}(d(v_i) - 1) \;\geq\; 3\textstyle\sum_{v \in F}(d(v) - 1) \;\geq\; 3(|E| - |V| + 1) \qquad (3.1)$$

In addition since every vertex of F lies after the first $3k$ vertices in the sequence we have that,

$$\sum_{i > 3k}(d(v_i) - 1) \;\geq\; \sum_{v \in F}(d(v) - 1) \;\geq\; (|E| - |V| + 1)$$

which combined with $|E| \geq 3|V|/2$ results in

$$\sum_{i=1}^{n}(d(v_i) - 1) \geq 4(|E| - |V| + 1) > 2|E| - |V|,$$

a contradiction. This establishes the lemma and the correctness of the algorithm. □

Since Step 5 of algorithm gives a $3k$ branching, we get:

Theorem 3.13 *Given a graph G on n vertices, and an integer parameter k, we can determine whether or not G has a feedback vertex set of size at most k in $O((3k)^k n + m)$ time.*

Note that the Lemma 3.11 leads to a $c^k m$ algorithm to determine whether a regular graph has a fvs of size at most k or not. It says that in a regular graph, for every $S \subseteq V$ with $|S| = 3k$ and for every fvs F with $|F| \leq k$, we have $F \cap S \neq \emptyset$. Hence a regular graph has a fvs of size k if and only if for every $S \subset V$ with $|S| = 4k$ and for every fvs F of size k, we have $F \subseteq S$. So given any regular graph G, the algorithm simply picks up some $4k$ vertices in the graph and tries all possible k-subsets of this as a possible fvs of size at most k, and if $n < 4k$, say NO.

Theorem 3.14 *There is an $O((4e)^k kn)$ algorithm to determine whether a given r-regular graph on n vertices has a feedback vertex set of size at most k.*

In Section 3.5, we obtained this result using a different approach. Note that the algorithm is *the same* though the proof of the correctness is not.

3.6.2 Bounds on girth

Now we look at a bound on the girth based on the maximum number of vertex disjoint cycles in the graph. This bound follows from the result obtained by Erdös and Posa in [93]. Let

$$\begin{aligned}\mathbb{G}(i,t) &= \{G \mid \; G \text{ is a graph with minimum degree} \\ &\qquad\quad i \text{ and has at most } t \text{ vertex disjoint cycles}\}\end{aligned}$$

It is shown in [93] that

$$\begin{aligned} g(i,t) &= \max_{G \in \mathbb{G}(i,t)} g(G) \\ &\leq 2 + 2\left(\frac{(1+o(1))}{\log(i-1)}\right)\log t \end{aligned}$$

An analysis of the above result found in [221] and quoted in [188] shows that the expression $(1+o(1))$ can be bounded by 4. So for any G with minimum degree 3, having at most t vertex disjoint cycles, the girth $g(G)$ is bounded by

$$g(G) = g(3,t) \leq 8\log t + 2$$

Since the size of any feedback vertex set is bounded below by the maximum number of vertex disjoint cycles of a graph, we have the following.

Lemma 3.12 *Let G be a graph with minimum degree 3 and having a feedback vertex set of size at most k then $g(G) \leq 8\log k + 2$.*

As a corollary, we get

Theorem 3.15 *Given a graph G on n vertices, and an integer parameter k, we can determine whether or not G has a feedback vertex set of size at most k in $O((8\log k + 2)^k n^{\omega})$.*

Note that the bound (on girth) obtained in Corollary 3.2 is better than the bound given here in Lemma 3.12.

3.7 Conclusion

In this chapter, we proved that graphs with minimum degree 3 having a small fvs possess short cycles. Using this we obtained faster algorithms for the FEEDBACK VERTEX SET problem on undirected graphs. Our main result achieves a significant improvement in the dependence on k (the parameter) of the running time. We get an algorithm with $O\left(\left(\frac{12\log k}{\log\log k}+6\right)^k n^{\omega}\right)$ running time.

A number of advances have been made on reducing the $f(k)$ for the FVS problem. Dehne et. al. [69] have obtained an algorithm for FVS problem that runs in time $O(c^k n^3)$, where $c = 10.567$. Independently, Guo et. al.[138] have also obtained a $O(c^k mn)$, where $c = 37.7$, time algorithm for the FVS problem.

Apart from its application to the design of FPT algorithms, our Lemma 3.9 and Theorems 3.2 and 3.5 may be of independent interest in extremal graph theory.

Theorem 3.5 essentially shows that the size of the problem kernel for the feedback vertex set problem is $O(k^{1+2\epsilon})$ for fixed $\epsilon \leq 1/2$. This is because we can reduce the problem size to $O(k^{1+2\epsilon})$ in $O((\frac{6}{\epsilon})^k n^\omega)$ time. We can use these kernels in connection with newly developed algorithms by first branching on cycles of length at most 6 and using the improved c^k algorithms only when girth of the graph exceeds 6 in which case the instance size is at most $O(k^2)$. This will give a fixed parameter tractable algorithm with time complexity $O(6^k n^\omega + (10.567)^k k^6)$ using the algorithm developed in [69]. Recently, combining branching and iterative compression Chen et. al. [52] gave an algorithm for FVS running in time $O(5^k n^{O(1)})$.

4 Feedback Set Problems in Directed Graphs

While we gave fixed parameter tractable algorithms for feedback set problems in undirected graphs in the last chapter (the edge version in undirected graphs can be trivially solved), the parameterized complexity of feedback set problems in directed graphs has been a long standing open problem in the area until recently [54]. In fact, there are problems on sequences and trees in computational biology, that are related to the directed feedback vertex set problem [102].

In this chapter we consider directed versions of feedback set problems in special classes of directed graphs. We also consider the dual problems of feedback set problems in general directed graphs. The problems explored in this chapter are

> Directed Feedback Vertex (Arc) Set (FVS (FAS)): Given a directed graph $G = (V, E)$ and an integer parameter $k \geq 0$, determine whether there exists a set of at most k vertices (arcs) whose removal results in an acyclic directed graph.
>
> Weighted Directed Feedback Vertex (Arc) Set (WFVS (WFAS)): Given a directed graph $G = (V, E)$, $\pi : V \rightarrow \Re^+$ ($\pi : E \rightarrow \Re^+$) and an integer parameter $k \geq 0$, determine whether there exists a set of vertices (arcs) of weight at most k, whose removal makes the graph acyclic.

We consider these problems in the well studied special class of directed graphs, tournaments. A tournament $T = (V, E)$ is a directed graph in which there is exactly one directed arc between every pair of vertices. The Feedback Vertex Set problem is known to be NP-complete in tournaments [214] but a NP-completeness proof for the Feedback Arc Set problem in tournaments eluded us until recently. Alon [8], Charbit et. al. [50] and Contizer [63] independently showed that Feedback Arc Set is NP-complete for unweighted tournaments in 2005. Earlier, FAS was only known to be NP-complete for weighted tournaments [86]. We give efficient fixed parameter tractable algorithms for the feedback vertex set and feedback arc set problems in weighted tournaments.

The Weighted Feedback Arc Set problem in tournaments finds application in rank aggregation methods. Dwork et. al. [86] have shown that the problem of computing the so called Kemeny optimal permutation for k full lists, where k is an odd integer, is reducible to the problem of computing a minimum feedback arc set problem on a weighted tournament with weights between 1 and $k-2$.

We first give two simple algorithms for the unweighted Feedback Vertex Set problem in tournaments in section 4.1. In Section 4.2, we give algorithms for the weighted version of the feedback vertex set problem in tournaments. We consider the following variants of the weighted feedback vertex set (WFVS) problem:

1. Integer-WFVS, where the weights are arbitrary positive integers,
2. Real-WFVS, where the weights are real numbers ≥ 1, and
3. General-WFVS, where the weights are positive real numbers.

We show that the Integer-WFVS in a directed graph can be solved as fast as the Feedback Vertex Set problem in an unweighted directed graph. Since the reduction here preserves the tournament structure, it follows that Integer-WFVS can be solved as fast as the feedback vertex set problem in an unweighted tournament, which currently has a running time of $O((2.27)^k + n^3)$ [190]. We show that Real-WFVS can be solved in $O((2.4143)^k n^\omega)$ time and that General-WFVS is not fixed parameter tractable unless $P = NP$.

There are parameterized reductions between the Feedback vertex set problem (FVS) and the Feedback Arc Set problem (FAS) in weighted directed graphs (actually the reductions used to show NP-completeness for these problems are parameterized reductions [95]), but they don't preserve the tournament structure. In Section 4.3 we give three different algorithms for the FAS problem.

Starting with a simple $O(\sqrt{k}^k n^\omega \log n)$ time algorithm based on branching on directed short cycles, we end up giving an $O((2.415)^k n^\omega)$ time algorithm based on a characterization of minimal feedback arc sets and branching on forbidden structures. We also observe that the algorithm, and hence the bound, applies for the FAS problem in weighted tournaments as well, where weights on the arcs are at least 1. In Section 4.4, we show that FAS is fixed parameter tractable even for dense directed graphs (graphs having at least $\binom{n}{2} - n^{1+o(1)}$ arcs).

In Section 4.5, we consider the parametric duals of feedback set problems in directed graphs. More specifically, the dual problems are : Given a directed graph G, (a) is there a set of at least k vertices of G that induces a directed acyclic graph, and (b) is there a directed acyclic subgraph of G with at least k arcs? We call the former problem V-MAXDAG and the later MAXDAG. In undirected graphs, the former ((a)) question is $W[1]$-complete [160] while the latter question is easily solvable in polynomial time (since in any connected graph on n vertices and m edges, it is necessary and sufficient to remove $m-(n+1)$ edges to make it acyclic).

In directed graphs where cycles of length 2 are allowed, we show that the V-MAXDAG is $W[1]$-hard, while it is fixed parameter tractable for oriented directed graphs (where cycles of length 2 are not allowed). We show that MAXDAG is fixed parameter tractable in general directed graphs. These algorithms use lower bound on the solution size. We also consider variations of these problems where the parameter is abovethe default lower bound.

Section 4.6 gives improved algorithms for MAXDAG and its variants based on an optimization version of MAXDAG and on kernelization. Here, we also show that the FAS is fixed parameter tractable when the graph has minimum out-degree or in-degree at least $f(n)$ (for some function of n).

We conclude with some remarks and discussions in Section 4.7.

Throughout this chapter, by $\log n$ and ω, we mean, respectively, the logarithm to the base 2 of n and the exponent of the running time of the best matrix multiplication algorithm. By $rev(x)$, where $x = (u, v)$ is an arc of a directed graph, we mean the arc (v, u). By an *oriented directed graph*, we mean a directed graph where there is at most one directed arc between every pair of vertices. By an *in-neighbor* of a vertex x in a directed graph G, we mean a vertex y such that there is a directed arc from y to x in G. An *out-neighbor* of a vertex is similarly defined. The *in-degree* $d^-(x)$ (*out-degree* $d^+(x)$) of a vertex x is the number of its in-neighbors (out-neighbors).

4.1 The Unweighted Feedback Vertex Set Problem in Tournaments

The starting point of our FPT algorithm for FVS in tournaments is the following lemma.

Lemma 4.1 ([18]) *A tournament $T = (V, E)$ has a directed cycle if and only if it has a directed triangle. A directed triangle can be found in $O(n^\omega)$ time.*

Lemma 4.1 implies that the feedback vertex set problem in a tournament is a set of vertices that hits all the triangles in the tournament. So one can first find a directed triangle in the tournament, and then branch on each of its three vertices to get an easy recursive $O(3^k n^\omega)$ algorithm to find a feedback vertex set of size at most k (or determine its absence).

Theorem 4.1 *Given a tournament $T = (V, E)$, we can determine whether T has a feedback vertex set of size at most k in $O(3^k n^\omega)$ time.*

Alternatively, we can formulate the unweighted feedback vertex set problem in tournaments as a 3-hitting set problem (the problem of hitting all directed triangles) and apply the hitting set algorithm of [190] to get

Theorem 4.2 *Given a tournament $T = (V, E)$, we can determine whether T has a feedback vertex set of size at most k in $O((2.27)^k + (kn)^3)$ time.*

The algorithm of Theorem 4.1 generalizes to the weighted feedback vertex set problem with weights at least 1 while the algorithm of Theorem 4.2 uses some preprocessing rules which don't naturally generalize to the weighted hitting set problem. In the next section, we give faster algorithms for the weighted feedback vertex set problem in tournaments than what we can get directly from the generalization of Theorem 4.1.

4.2 The Weighted Feedback Vertex Set Problem in Tournaments

In this Section we look at various weighted versions of feedback vertex set problem in tournaments.

4.2.1 Integer-WFVS

Integer-WFVS is a variant of the weighted feedback vertex set problem, where weights are arbitrary positive integers.

Theorem 4.3 *There exists a parameterized many-one reduction from* Integer-WFVS *to the unweighted feedback vertex set problem in directed graphs.*

Proof: Let G be an integer weighted directed graph. Any vertex having weight strictly more than k can not be a part of any minimal feedback vertex set of weight at most k. So, given the weight function π, if some vertex v has $\pi(v) > k$ then we make $\pi(v) = k + 1$. It is easy to see that G has a feedback vertex set of weight at most k if and only if it has a feedback vertex set of weight at most k with the modified weight function.

We will construct a new directed graph G' from G as follows: replace each vertex v having weight $\pi(v) = w > 1$ with a cluster V' consisting of w vertices. If there is an arc (u, v) in the original graph G then we add an arc from every vertex of the cluster U' to every vertex in V'. For every cluster V', we add intra cluster arcs such that $G[V']$ is an acyclic tournament. Here $G[V']$ represents the induced directed graph on V'.

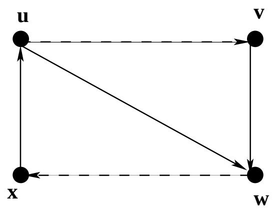

Figure 4.1: A Witness Cycle

We claim that G has a feedback vertex set of weight at most k if and only if G' has a feedback vertex set of size at most k. Let $\{v_1, v_2, \cdots, v_l\}$ be a FVS of weight at most k in G. Then the vertices of the corresponding clusters $\{V_1', V_2', \cdots, V_l'\}$ form a fvs of size at most k in G'. The other direction follows from the observation that every minimal feedback vertex set (F) of size at most k in G' has either all the vertices of any cluster or none of them. To see this, assume that there is a cluster V' such that there is a vertex $v \in V'$ in F and a $u \in V'$ not in F (see Figure 4.1). Since $v \in F$, and F is minimal, there exists a witness cycle C such that $F \cap C = \{v\}$. Now if $u \notin C$ then we get a cycle C' in T' by replacing v with u in C such that $C' \cap F = \emptyset$ contradicting the definition of F. If u is part of this cycle then the length of the cycle is at least 4. Let C be $\{u, \cdots, v, w, \cdots, x\}$ and construct C' as $\{u, w, \cdots, x\}$. Then $C' \cap F = \emptyset$ a contradiction. This proves the other direction. The number of vertices in the new instance of the graph is bounded by $(k+1)n$ and this instance can be obtained in polynomial time from G. □

Corollary 4.1 Integer − WFVS *in tournaments can be solved in* $O((2.27)^k + (kn)^3)$ *time.*

Proof: Let T' be the graph obtained from T by applying Theorem 4.3. Then clearly T' is a tournament if T is. Now the corollary follows from Theorem 4.2. □

4.2.2 Real- and General-WFVS

If the weights are arbitrary reals, but at least 1, then the algorithm for unweighted tournament can not be directly applied. Here we give an algorithm which attains a bound of $O((2.4143)^k n^\omega)$.

Let M be the adjacency matrix of an oriented directed graph T. Then T has a directed triangle if and only if for some i, j such that $1 \leq i < j \leq n$, $M^2[i,j] \geq 1$ and $M[j,i] = 1$. This can be determined in $O(n^\omega)$ time. If such a pair (i,j) exists, then there exists a k such that $M[i,k] = M[k,j] = 1$ which can also be determined in $O(n)$ time. Such a triple $\{i,j,k\}$ forms a triangle. Further, T has a directed cycle of length 4 if and only there exists a pair (i,j) such that $1 \leq i < j \leq n$, $M^2[i,j] \geq 1$ and $M^2[j,i] \geq 1$. If such a pair exists, then as before, the witness 4-cycle can also be found in $O(n)$ time. So we have

Lemma 4.2 *Let T be an oriented directed graph on n vertices. Then we can find a directed triangle or a directed cycle of length 4, if it exists, in $O(n^\omega)$ time.*

We need the following lemma for our algorithm.

Lemma 4.3 *Let $T = (V, A)$ be a weighted tournament that does not contain a directed cycle of length 4. Then the minimum weight feedback vertex and arc set problems are solvable in T in $O(n^\omega)$ time.*

Proof: It is easy to see that if a tournament does not have a directed cycle of length 4 then no pair of directed triangles in the tournament has a vertex in common. Hence the minimum weight feedback vertex or arc set is obtained by finding all triangles, and picking a minimum weight vertex/arc from each of them.

Finding all triangles in such a tournament can be done in $O(n^\omega)$ time as follows. First compute M^2, the square of the adjacency matrix of the tournament. Since the tournament can have at most $n/3$ triangles, there can be at most $n/3$ pairs (i, j) such that $1 \leq i < j \leq n$ and $M^2[i, j] \geq 1$ and $M[j, i] = 1$. For each such pair, the corresponding witness triangle can be found in $O(n)$ time. $\square$

We remark that a tournament T has a directed cycle of length 4 if and only if it has a subgraph isomorphic to F_1 (see Figure 4.2). To see this, it is enough to observe that given a directed cycle of length 4, in any orientation of its diagonals there always exists an arc on this 4 cycle such that the heads of these diagonals are the endpoints of this arc. Hence by Lemma 4.2, F_1 can be found in $O(n^\omega)$ time. This gives us the following lemma.

Lemma 4.4 *Let $T = (V, A)$ be a tournament then T has a directed cycle of length 4 if and only if it has a subgraph isomorphic to F_1 (see Figure 4.2) and F_1 can be found in $O(n^\omega)$ time.*

Algorithm TFVS(T, k, π, F)(*T is a tournament, $k \geq 0$, π is a weight function on V, F is a set of vertices*)
(Returns 'true' and a minimal feedback vertex set of weight at most k, if one exists and returns 'no' otherwise. F contains vertices of a partial feedback vertex set that are deleted from the original T. Initially the algorithm is called by TFVS$(T, k, \pi, \emptyset)$.)

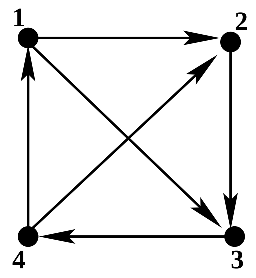

Figure 4.2: F_1

Step 0: If $k = 0$ and T has a triangle, then answer 'no' and exit.

Step 1: If T does not have a directed triangle and $k \geq 0$, then return 'true' and F and exit.

Step 2: Find an induced subgraph on 4 vertices isomorphic to F_1 (as in Figure 4.2) with vertex set, say $\{1, 2, 3, 4\}$ and with adjacencies as in Figure 4.2. If no such subgraph exists, then go to Step 4.

Step 3: If any of the following recursive calls results in true, then return 'true' and the corresponding F and exit, else return 'no' and exit. In the following, T' is obtained by deleting the 'newly included' vertices in F.

1. $TFVS(T', k - \pi(3), \pi, F \cup \{3\})$,
2. $TFVS(T', k - \pi(4), \pi, F \cup \{4\})$,
3. $TFVS(T', k - \pi(1) - \pi(2), \pi, F \cup \{1, 2\})$

Step 4: Find a minimum weight feedback vertex set S for the resultant tournament using Lemma 4.3 in polynomial time. If $\pi(S) > k$ then return 'no' and exit, else return 'true' and $F \cup S$ and exit. Here $\pi(S) = \sum_{v \in S} \pi(v)$.

The correctness of Steps 0 and 1 follows from Lemma 4.1. In Step 3, we branch on all possible minimal solutions of F_1. Step 4 follows from Lemma 4.3.

Since the weight of each vertex is at least 1 we reduce k by at least 1 in first two branches of the recursion and at least by 2 in the last branch. Hence the time taken by the algorithm is bounded by the following recurrence:

$$T(n, k) \leq 2T(n - 1, k - 1) + T(n - 2, k - 2) + O(n^{\omega})$$

which solves to $O((2.4143)^k n^\omega)$. So we have the following theorem.

Theorem 4.4 *Given a tournament $T = (V, E)$, and a weight function $\pi : V \rightarrow \Re^+$, such that $\pi(v)$ is at least 1 for every $v \in V$, we can determine whether T has a feedback vertex set of weight at most k in $O((2.4143)^k n^\omega)$ time.*

The General-WFVS problem, where the weights can be arbitrary positive reals, is not fixed parameter tractable unless $P = NP$. We show this by proving that it is NP-complete for some fixed constant k (in fact, for $k = 1$). Our reduction is from the NP-complete unweighted feedback vertex set problem in tournaments [214]. Let T be a tournament on n vertices where we are interested in finding a FVS of size k'. Define the weight function π to be $\pi(v) = 1/k'$ for all $v \in V$. Then the original tournament has a FVS of size k' if and only if the resulting weighted tournament has a FVS of weight 1. This implies that there cannot be a $f(k)n^{O(1)}$ or even an $n^{O(k)}$ time algorithm for General-WFVS problem unless $P = NP$. This result is true for general directed graphs since the unweighted feedback arc set problem is NP-complete for general directed graphs.

Theorem 4.5 *The General-WFVS problem is not fixed parameter tractable in general directed graphs unless $P = NP$.*

4.3 The Feedback Arc Set Problem in Tournaments

In the last Section we gave fixed parameter tractable algorithms for the feedback vertex set problem in tournaments. It is not straightforward to apply the ideas of the fixed parameter tractable algorithms for the feedback vertex set problem to the arc set problem as it is not sufficient to hit all triangles by arcs to get a feedback arc set in a tournament (for example, in the tournament of Figure 4.2, $S = \{(1,3), (4,2)\}$ hits all the triangles, but S does not hit the cycle $\{1, 2, 3, 4\}$). Furthermore, after we remove an arc from a tournament, we no longer have a tournament. In this Section we give three different algorithms for the feedback arc set problem for tournaments. We first develop an $O(\sqrt{k}^k n^\omega \log n)$ time algorithm for FAS using the fact that a directed graph G with at most k arcs away from a tournament T (i.e., T can be obtained from G by adding at most k arcs to it) has a cycle of length at most $O(\sqrt{k})$. In subsection 4.3.2, we first show that if a subset F of arcs forms a minimal feedback arc set in a directed graph then the graph formed after reversing these arcs is acyclic. Such a characterization helps us to maintain the tournament structure (since in every recursive step we reverse but do not delete arcs). We apply this characterization to develop an algorithm for FAS in tournaments taking $O(3^k n^\omega)$ time. We then improve this by using a branching technique to obtain an $O((2.415)^k n^\omega)$ time algorithm.

4.3.1 The Feedback Arc Set Problem in Tournaments is FPT

We will first obtain a bound on the length of a shortest cycle in a graph with at most k arcs away from a tournament.

Lemma 4.5 *Let $G = (V, E)$ be a directed graph such that $|V| = n$ and $|E| \geq \binom{n}{2} - k$, for some non-negative integer k. Then either G is acyclic or has a directed cycle of length at most $c\sqrt{k}$ for some positive constant c.*

Proof: Assume G is not acyclic and choose c such that $c^2k - 3c\sqrt{k} \geq 2k$ ($c = 3$ suffices for $k \geq 2$). Note that the shortest directed cycle C of G is chordless; i.e. for all non-adjacent pairs of vertices u, v in C, there is no arc (u, v) or (v, u) since otherwise that arc between u and v will give rise to a shorter directed cycle. Suppose that the length l of the shortest directed cycle C in G is strictly greater than $c\sqrt{k}$. Then G does not have any chord of C and hence does not have at least $\binom{l}{2} - l = l(l-3)/2 > (c^2k - 3c\sqrt{k})/2 \geq k$ arcs. This is a contradiction since G has at least $\binom{n}{2} - k$ arcs. □

Now we are ready to show the following theorem.

Theorem 4.6 *Given a tournament $T = (V, E)$, we can determine whether it has a feedback arc set of size at most k in $O((c\sqrt{k/e})^k n^\omega \lg n)$ time, where c is a positive constant. I.e. the feedback arc set problem is fixed parameter tractable in tournaments. (Here e is the base of the natural logarithm function.)*

Proof: We give an algorithm *TFES* which constructs a search tree for which each node has at most $c\sqrt{k}$ children. Each node in the tree is labeled with a set of vertices S that represents a partially constructed feedback arc set.

Algorithm TFES($T = (V, E)$, k, F) (* $k \geq 0$ *)
(Returns 'true' and a feedback arc set of size at most k, if one exists and returns 'no' otherwise. F contains the arcs of a partial feedback arc set. Initially the algorithm is called by TFES($T, k, \emptyset$).)

Step 1: Find a shortest cycle C in T, if one exists.

Step 2: If T is acyclic, then return 'true' and $\emptyset$ and exit.

Step 3: If $k = 0$, then answer 'no' and exit.

Step 4: If for some arc $e \in C$, TFES($T', k-1, F \cup \{e\}$) is true, where $T' = (V, E - e)$ then return 'true' and F (Note: Here F is actually $F \cup \{e\}$.) and exit, else answer 'no' and exit.

If the algorithm exits at Step 2, then G can be made acyclic by not deleting any arc and hence its answer is correct (since $k \geq 0$). If it exits at Step 3, then G has a cycle and so it can't be made acyclic by deleting $k = 0$ arcs, and so its answer is correct. Finally the correctness of Step 4 follows from the fact that any feedback arc set must have one of these arcs of the cycle C, and the step is recursively checking for each arc e in the cycle whether $T - e$ has a feedback arc set of size at most $k - 1$.

To show that the algorithm takes the claimed time bounds, observe that since k decreases at every recursive Step 4 (after an edge deletion), the recursion depth is at most k. Also the resulting directed graph after the i-th step of the recursion has at most i arcs deleted from a tournament. Hence Lemma 4.5 applies and so there is a cycle of length at most $c\sqrt{i}$ in the resulting graph after the i-th step. So the number of nodes in the search tree is $O(c^k\sqrt{k!})$. The shortest cycle in a directed graph can be found in $O(n^{\omega} \lg n)$ time [149]. Hence the claimed time bound follows from Stirling's approximation. □

4.3.2 Improved Algorithms

The algorithms in this Section are based on Lemma 4.6 which was observed independently by Gallai [143] and Grinberg et al. [136]. We state it and give a proof here for completeness.

Lemma 4.6 (Reversal Lemma)
Let $G = (V, E)$ be a directed graph and F be a minimal feedback arc set (FAS) of G. Let G' be the graph formed from G by reversing the arcs of F in G. Then G' is acyclic.

Proof: Assume to the contrary that G' has a cycle C. Then C can not contain all the arcs of $E - F$, as that will contradict the fact that F is a FAS. Define the set $rev(F) = \{(u, v) \mid (v, u) \in F\}$. Let $C \cap rev(F) = \{f_1, f_2, \cdots, f_k\}$ and $e_i = rev(f_i)$. Then the set $\{e_1, e_2, \cdots, e_k\}$ is a set of arcs of G which are reversed and are part of C. Now since each $e_i \in F$, and F is minimal, there exists a cycle C_i in G such that $F \cap C_i = \{e_i\}$. Now consider the directed graph L induced by the arcs of $\{C, C_1, \cdots, C_k\} - F - rev(F)$. It is clear that L is a directed closed walk with all the arcs in the original graph G. In fact, if $\forall i,\ C_i \cap C = \emptyset$, then L is a simple cycle in G, such that $L \cap F = \emptyset$, contradicting the fact that F is a FAS. If L is not a simple cycle then we can extract a simple directed cycle from it not having any arcs of F, violating the definition of F. □

We use Lemma 4.6 to give an improved algorithm for the feedback arc set problem in a tournament.

Algorithm TFAS(T,k, F)(*T is a tournament, $k \geq 0$, and F is a set of arcs.*) (Returns 'true' and a minimal feedback arc set of size at most k, if one exists and returns 'no' otherwise. F contains the arcs of a partial feedback arc set that are reversed from the original T. Initially the algorithm is called by TFAS$(T, k, \emptyset)$.)

Step 0: If T does not have a directed triangle and $k \geq 0$, then return 'true' and F.

Step 1: If $k = 0$ and T has a triangle, then answer 'no'.

Step 2: Find a triangle in T and let $\{a, b, c\}$ be the arcs of the triangle.

Step 2a: If $rev(a), rev(b)$ and $rev(c)$ are in F, then answer 'no' and exit.

Step 2b: If $TFAS(T\backslash\{x\} \cup rev\{x\}, k - 1, F \cup \{x\})$ is true for any arc x of the triangle such that $rev(x)$ is not in F, then return 'true' and F (Note: Here F is actually $F \cup \{x\}$.) and exit. Otherwise return 'no' and exit.

Theorem 4.7 *Given a tournament $T = (V, E)$ on n vertices, we can determine whether it has a feedback arc set of size at most k in $O(3^k n^\omega)$ time.*

Proof: First we will show that the algorithm $TFAS$ finds a minimal feedback arc set of size at most k if one exists. Correctness of Step 0 and Step 1 follow from Lemma 4.1. Step $2a$ answers correctly as by Reversal Lemma, the current F can not be extended to a minimal feedback arc set of G. In Step $2b$, we branch on each arc x of the triangle such that $rev(x) \notin F$, because if none of these arcs is picked in the feedback arc set of G, then this triangle will survive in G', obtained by reversing the arcs of F. But then by Reversal Lemma, this F is not minimal. So this proves the correctness of the algorithm.

The claimed time bound can easily be seen by observing that k decreases at every recursive Step $2b$ by 1. So the recursion depth is at most k. The branching factor at every recursion step is at most 3 and hence by Lemma 4.2, we have the desired time bound for the algorithm. □

We further improve the time bound using a better branching technique.

Algorithm BTFAS(T,k, F)(*T is a tournament, $k \geq 0$, F is a set of arcs*) (Returns 'true' and a minimal feedback arc set of size at most k, if one exists and returns 'no' otherwise. F contains the arcs of a partial feedback arc set that are reversed from the original T. Initially the algorithm is called by BTFAS$(T, k, \emptyset)$.)

Step 0: If T does not have a directed triangle, then return 'true' and F.

Step 1: If $k = 0$ and T has a triangle, then answer 'no' and exit.

Step 2: Find an induced subgraph on 4 vertices isomorphic to F_1 (as in Figure 4.2), if exists, in T. Such a subgraph is simply a tournament on 4 vertices having at least two directed triangles. Let the vertex set of such an F_1 be $\{1, 2, 3, 4\}$ and the adjacencies be as in Figure 4.2 (in particular $(1, 2)$ is the only arc not part of any directed triangle). If no such subgraph exists in T, then go to Step 6.

Step 3: Let $\{a, b, c\}$ be the arcs of a triangle in F_1, such that there exists an arc $x \in \{a, b, c\}$ for which $rev(x) \in F$. If there is no such triangle in F_1, then go to Step 4.

Step 3a: If $rev(a), rev(b)$ and $rev(c)$ are in F, then answer 'no' and exit.

Step 3b: If $BTFAS(T \backslash \{x\} \cup rev(x), k-1, F \cup \{x\})$ is true for any arc x of the triangle such that $rev(x)$ is not in F, then return 'true' and F (Note: Here F is actually $F \cup \{x\}$.) and exit; else answer 'no' and exit.

Step 4: If $rev((1,2)) \notin F$ then if any of the following recursive calls returns true, then return 'true' and the corresponding F and exit, and answer 'no' and exit otherwise.

In the following, T' is obtained from T by reversing the 'newly included' arcs of F.

1. $BTFAS(T', k-1, F \cup \{(3,4)\})$,
2. $BTFAS(T', k-2, F \cup \{(4,1),(4,2)\})$,
3. $BTFAS(T', k-2, F \cup \{(4,1),(2,3)\})$,
4. $BTFAS(T', k-2, F \cup \{(1,3),(2,3)\})$,
5. $BTFASF(T', k-3, F \cup \{(1,2),(1,3),(4,2)\})$

Step 5: If $rev((1,2)) \in F$, then if any of the first 4 recursive calls enumerated in Step 4 returns true, then return 'true' and the corresponding F and exit, and answer 'no' otherwise.

Step 6: Find a minimum feedback arc set S of the resultant tournament using Lemma 4.3 in polynomial time. If $|S| > k$ then return 'no' and exit else return 'true' and $F \cup S$ and exit.

In the above algorithm at every step, we first find a graph isomorphic to F_1, and then if there exists a directed triangle in F_1 with all its arcs included in the partial feedback arc set (F) obtained so far, then we answer 'no' which is justified by Lemma 4.6. Otherwise we branch on all the arcs x of the triangle such that $rev(x) \notin F$ as by Lemma 4.6 at least one such arc must be part of F.

If none of the arcs of F_1 is part of F, then we branch on all possible minimal feedback arc sets of F_1. The only remaining case is when all the arcs x appearing in some triangle in F_1 are not in F but $rev((1,2)) \in F$. In this case, Lemma 4.6 implies that item 5 of *Step 2b* is not applicable (because the set $\{(1,3),(4,2)\}$ is not a minimal FAS of F_1). So when we reach *Step* 6 of the above algorithm, all the induced subgraphs on 4 vertices have at most one triangle. And the problem now can be solved in polynomial time by Lemma 4.3 .

Thus, we get the following recurrence for the time complexity of the algorithm:

$$T(n,k) \leq \max \begin{cases} 2T(n,k-1) + O(n^\omega) \text{ or} & \text{(Step 3b)} \\ T(n,k-1) + 3T(n,k-2) + T(n,k-3) + O(n^\omega) & \\ & \text{(Step 4)} \end{cases}$$

The above recurrences solve to $O((2.415)^k n^\omega)$. So we get the following theorem.

Theorem 4.8 *Given a tournament $T = (V,E)$, we can determine whether it has a feedback arc set of size at most k in $O((2.415)^k n^\omega)$ time.*

The above algorithm primarily depends on the facts that when we include an arc in partially constructed feedback arc set during recursion then we decrease k by 1 and when the tournament does not have an F_1 we can solve the problem in polynomial time using Lemma 4.3.

Hence we note that the above algorithm can also be applied to the weighted feedback arc set problem for tournaments where the weight of every arc is at least 1 as both branching rules and Lemma 4.3 are applicable for this version of the weighted case.

Theorem 4.9 *Given a tournament $T = (V,E)$, and a weight function $\pi : E \rightarrow \Re^+$, such that $\pi(e)$ is at least 1 for every $e \in E$, we can determine whether T has a feedback arc set of weight at most k in $O((2.415)^k n^\omega)$ time.*

We conclude this Section with the following theorem which shows that the weighted version of the feedback arc set problem, where weights can be arbitrary, is unlikely to be fixed parameter tractable unless $P = NP$. We call this version of the weighted feedback arc set problem General-WFAS. The proof of the next theorem is similar to the proof of Theorem 4.5 and follows from a reduction from the unweighted version of the feedback arc set problem, which is known to be NP-complete.

Theorem 4.10 *General-WFAS problem is not fixed parameter tractable in general directed graphs unless $P = NP$.*

4.4 Feedback Arc Set Problem in Dense Directed Graphs

In this section, we show that the feedback arc set problem is fixed parameter tractable for directed graphs which are at most $n^{1+o(1)}$ arcs away from a tournament. We need the following lemma to show the desired result. Recall that the girth of a graph is defined as the length of the shortest cycle in the graph. A directed graph is called *strongly connected* if there exists a directed path between every pair of vertices.

Lemma 4.7 *[23] Let $G = (V, E)$ be a strongly connected directed graph with n vertices, m arcs and let $l \geq 2$. Then if $m \geq \frac{n^2+(3-2l)n+(l^2-l)}{2}$, the girth of the graph $(g(G))$ is bounded by l.*

Corollary 4.2 *Let G be a strong directed graph with n vertices and $m \geq \binom{n}{2} - \frac{n(g-2)}{2}$ where $3 \leq g \leq n-6$. Then $g(G) \leq g$.*

Proof: We first note that

$$\binom{n}{2} - \frac{n^2+(3-2g)n+(g^2-g)}{2} = \frac{2n(g-2)+g-g^2}{2}$$

and

$$\frac{2n(g-2)+g-g^2}{2} \geq n(g-2) - \frac{g^2}{2} \geq \frac{n(g-2)}{2} \quad \text{whenever } n \geq \tfrac{g^2}{g-2}\ . \tag{4.1}$$

To show $n \geq \frac{g^2}{g-2}$, it suffices to show

$$\begin{aligned} n &\geq \frac{g^2}{g-2} \\ &= \frac{g^2-4}{g-2} + \frac{4}{g-2} \\ &= g+2+\frac{4}{g-2}. \end{aligned}$$

We know $g+2+\frac{4}{g-2} \leq g+6 \leq n$, which completes the claim.

Now Lemma 4.7 in connection with inequality 4.1 implies that if a strongly connected directed graph has at least $\binom{n}{2} - \frac{n(g-2)}{2}$ arcs, then its girth is bounded by g. □

Theorem 4.11 *Let G be a directed graph with n vertices and $m \geq \binom{n}{2} - n^{1+o(1)}$ arcs. Then the feedback arc set (FAS) problem is fixed parameter tractable for G.*

Proof: For the feedback arc set problem, we can assume without loss of generality, that the given directed graph is a strongly connected directed graph. (Otherwise, try values up to k in each strongly connected subgraph and take the minimum.)

We find the shortest cycle in G and then by applying Lemma 4.6, we branch on each arc by reversing the arc. This way we don't delete any arc and hence at every recursive step Corollary 4.2 ensures a cycle of length at most $n^{o(1)}$. So we have an algorithm for feedback arc set problem in G which takes $O((n^{o(1)})^k n^{O(1)})$ time. Cai and Judes [45] have observed that an $O((n^{o(1)})^k)$ algorithm can be simulated by an algorithm of time $f(k)n^{O(1)}$, where f is some function of k, for every fixed n and k. Hence it follows that the feedback arc set problem is fixed parameter tractable for G. □

Note that the proof does not carry over to the FVS problem on dense directed graphs. This is because, we may not obtain a dense directed graph after deleting a vertex from a dense directed graph.

4.5 Parametric Duals

The parametric dual of a parameterized problem with parameter k is the same problem with k replaced by 'all but k' ([160, 177, 178]). For example, the parametric dual of the k-vertex cover is the $(n-k)$- vertex cover or equivalently the k-independent set problem.

In this section, we show that the parametric dual problems of the directed feedback set problems are themselves some natural optimization problems and their parameterized versions are fixed parameter tractable in oriented directed graphs not just in tournaments.

4.5.1 The Parametric Dual of Directed Feedback Vertex Set - VMAXDAG

The parametric dual of the directed feedback vertex set problem is :

> Dual of Directed Feedback Vertex Set (V-MAXDAG): Given a directed graph on n vertices, are there at most $n-k$ vertices whose removal makes the graph acyclic? Or equivalently, is there a set of at least k vertices that induces an acyclic directed graph?

Given a tournament T, we can find a subset S of vertices such that $|S| \geq \lfloor \lg n \rfloor$ and the induced subtournament of T on S is acyclic (transitive). We *repeatedly* include the vertex with the smallest in-degree in the given tournament into S and remove it and its inneighbors from the tournament. It is also clear that the induced subtournament on S is acyclic. For, if we order the vertices by the order in which they are included in S, then the arcs go only from smaller vertices to bigger vertices.

To show that $|S| \geq \lfloor \lg n \rfloor$, it suffices to show that the 'repeat' loop will execute for at least $\lfloor \lg n \rfloor$ steps. This follows because in any tournament there is a vertex with in-degree at most $(n-1)/2$. Thus, after one step of the loop at most $(n+1)/2$ vertices are deleted.

Any oriented directed graph (without directed cycles of length 2) can be completed to a tournament by adding the missing arcs (with arbitrary directions). Hence every oriented directed graph G on n vertices and m arcs has at least $\lfloor \lg n \rfloor$ vertices that induce an acyclic subgraph. If the subgraph contains the newly added edges, just delete them.

Let the oriented directed graph G be given as an adjacency list where associated with every vertex x is a list of vertices y such that y is an in-neighbor of x. It is easy to implement each step of the 'repeat' loop in $O(m)$ time (We can have a bit-vector for the list of vertices to be deleted and scan through the adjacency list and remove those vertices.). By exiting the loop when $|S| = \lfloor \lg n \rfloor$, we get the following lemma.

Lemma 4.8 *Let G be an oriented directed graph with n vertices and m arcs. Then there exists a subset of $\lfloor \lg n \rfloor$ vertices which induces an acyclic subgraph and it can be found in $O(\min\{m \lg n, n^2\})$ time.*

Now we can design a fixed parameter tractable algorithm for the parameterized V-MAXDAG problem as follows. If $k \leq \lfloor \lg n \rfloor$, then return the acyclic subgraph obtained in Lemma 4.8, otherwise $n \leq 2^k$ and then we check all k sized subsets of the vertex set to see whether the subset induces an acyclic subgraph. If any one of them does, then we return the acyclic subgraph, otherwise answer 'no'. Since $\binom{n}{k} \leq \binom{2^k}{k} \leq (e2^k/k)^k$, we have the following theorem. (Here e is the base of natural logarithm.)

Theorem 4.12 *Given an oriented directed graph G and an integer k, we can determine whether or not G has at least k vertices that induce an acyclic subgraph in time $O((e2^k/k)^k k^2 + min(\{m \lg n\}, \{n^2\}))$; i.e V-MAXDAG problem is fixed parameter tractable.*

However it turns out that V-MAXDAG problem is $W[1]$-hard in general directed graphs.

Theorem 4.13 *It is $W[1]$-hard to determine whether a given directed graph has k vertices that induce an acyclic subgraph; i.e. V-MAXDAG problem is $W[1]$-hard in directed graphs.*

Proof: We reduce the k-independent set problem in undirected graphs to the given problem. Given an undirected graph $G = (V, E)$, an instance of the independent set problem we construct $D = (V, E^{'})$, an instance of V-MAXDAG problem in directed graph by adding both arcs $u \rightarrow v$ and $v \rightarrow u$ for every (u, v) in E. If G has an independent set of size k, then those corresponding vertices of D form an acyclic subgraph. Conversely, if D has an acyclic subgraph on k vertices, then those k vertices must form an independent set in D as if there is an arc between a pair of vertices in D, then there actually is a directed cycle (of length 2) between them. $\square$

The fixed parameter tractable algorithm for the V-MAXDAG problem follows from the easy observation that there is a "guarantee" (lower bound) of $\lfloor \lg n \rfloor$ for the solution size. In such situations, it is natural to parameterize above the guarantee [177] and so a natural question to ask is whether a given directed graph has a set of at least $\lfloor \lg n \rfloor + k$ vertices that induces an acyclic subgraph. The parameterized complexity of this question is open.

4.5.2 The Parametric Dual of Directed Feedback Arc Set - MAXDAG

The parametric dual of the directed feedback arc set problem is :

> Dual of Directed Feedback Arc Set (MAXDAG): Given a directed graph on n vertices and m arcs, are there at most $m - k$ arcs whose removal makes the graph acyclic. Or equivalently, is there a set of at least k arcs that induces an acyclic directed graph?

We show that MAXDAG is fixed parameter tractable which follows from the following easy lemma.

Lemma 4.9 *[18] Given a directed graph G on n vertices and m arcs, there always exists a set of at least $\lceil m/2 \rceil$ arcs that form an acyclic directed graph. Such a set of arcs can be found in $O(m)$ time.*

Proof: Order the vertices of the directed graph G arbitrarily. If m is the number of arcs in the graph, then at least $\lceil m/2 \rceil$ of these arcs go in one direction (all from a smaller vertex to a higher vertex or vice versa). These arcs form an acyclic directed graph. □

Theorem 4.14 *Given a directed graph G on n vertices and m arcs and an integer parameter k, we can determine whether or not G has at least k arcs that form an acyclic subgraph in time $O(4^k k + m)$, i.e MAXDAG problem is FPT in directed graphs.*

Proof: If $k \leq m/2$, then return the acyclic subgraph obtained in Lemma 4.9, otherwise $m \leq 2k$, and then check all k subsets of the arc set of the graph. If any of these k subsets of arcs induces an acyclic subgraph, then return the acyclic subgraph, and answer 'no' otherwise.

Since $\binom{m}{k} \leq 2^m \leq 2^{2k}$, and we can check in $O(k)$ time if k arcs forms an acyclic graph, we have the desired running time. □

Just as in the parametric dual of the feedback vertex set problem, a natural parameterized question here is whether the given directed graph has a set of at least $\lceil m/2 \rceil + k$ arcs that form an acyclic subgraph. This question is open for general directed graphs. In fact, $\lceil m/2 \rceil$ is a tight lower bound for the solution size in directed graphs. This bound is realized, for example, in a directed graph obtained by taking an undirected path on n vertices (and n edges) and replacing every edge by a pair of directed arcs one in each direction.

However, this bound of $\lceil m/2 \rceil$ is not tight for oriented directed graphs.

We prove that there exists an acyclic subgraph on $\frac{m}{2} + 1/2\lceil (n-c)/2 \rceil$ arcs in any oriented directed graph and then use this to give a fixed parameter algorithm for the question of 'whether the given oriented directed graph has a set of at least $\frac{m}{2} + k$ arcs that forms an acyclic subgraph'. We call this problem *above guarantee MAXDAG*.

We mimic the proof of the following lemma proved in [195].

Lemma 4.10 ([195]) *If G is a simple undirected graph (without parallel edges and self loops) with m edges, n vertices and c components, then the maximum number of edges in a bipartite subgraph of G is at least $\frac{m}{2} + \frac{1}{2}\lceil (n-c)/2 \rceil$. Such a bipartite graph can be found in $O(n^3)$ time.*

If we just apply Lemma 4.10 on the underlying undirected graph then at least half the arcs of the bipartite subgraph returned by the Lemma 4.10 are in one direction and that gives us a lower bound of $\frac{m}{4} + \frac{1}{4}\lceil (n-c)/2 \rceil$ on the size of maximum acyclic subgraph of G. However, by modifying the proof of Lemma 4.10, we get a bound of $\frac{m}{2} + \frac{1}{2}\lceil (n-c)/2 \rceil$. We will use $a(G)$ to denote the size of a maximum acyclic subgraph of G.

Lemma 4.11 *Any oriented directed graph $G = (V, E)$ with m arcs and n vertices, with the underlying undirected graph having c components, has an acyclic subgraph with at least $\frac{m}{2} + 1/2\lceil (n-c)/2 \rceil$ arcs, i.e. $a(G) \geq \frac{m}{2} + 1/2\lceil (n-c)/2 \rceil$ and such a subgraph can be found in $O(n^3)$ time.*

Proof: Without loss of generality assume that the underlying undirected graph is connected, otherwise we will apply this lemma on each component to get the result. The proof is along the lines of the proof of Lemma 4.10. We give the proof for completeness.

The proof is by induction on the number of vertices. The lemma is clearly true for oriented directed graphs on 1 or 2 vertices. At the induction step, there are three cases.

Case 1: The underlying undirected graph has a cut vertex x.

In this case, we apply induction on each of the connected components of the underlying undirected graph of $G - x$ including x in each component. Let the connected components of the underlying undirected graph of $G - x$ be $\{C_1, C_2, ..., C_k\}$ and let G_i denote the induced subgraph on $C_i \cup \{x\}$. Then we have:

$$a(G) \geq \sum_{i=1}^{k} a(G_i).$$

Let $E(G_i)$ denote the edge set of G_i and let m_i and n_i denote the cardinality of edges and vertices of G_i respectively. Observe that $\sum_i m_i = m$ and $\sum_i (n_i - 1) = n - 1$. Then by applying the induction hypothesis on G_i, we get

$$\begin{aligned} a(G) &\geq \sum_{i=1}^{k} \frac{m_i}{2} + \frac{1}{2}\lceil (n_i - 1)/2 \rceil \\ &\geq \frac{m}{2} + \frac{1}{2}\lceil (n-1)/2 \rceil \\ &\quad \text{(Since } \lceil x \rceil + \lceil y \rceil \geq \lceil x + y \rceil \text{, for any two rationals } x \text{ and } y.) \end{aligned}$$

Case 2: The underlying undirected graph has no cut vertex and the oriented directed graph G has a vertex x whose in-degree and out-degree are not the same.

In this case we apply induction on $G - x$ (whose underlying undirected graph will be clearly connected), and also include all arcs coming into x or all arcs going out of x whichever set is larger, into the acyclic subgraph to get an acyclic subgraph in the resulting directed graph. So we get:

$$\begin{aligned} a(G) &\geq a(G - x) + \frac{d_G(x) + 1}{2} \\ &\geq \frac{m - d_G(x)}{2} + \frac{1}{2}\lceil (n-2)/2 \rceil + \frac{d_G(x) + 1}{2} \\ &\geq \frac{m}{2} + \frac{1}{2}\lceil (n-1)/2 \rceil. \end{aligned}$$

Here $d_G(x)$ represents the number of neighbors (both in-neighbors and out-neighbors) of x in G.

Case 3: Every vertex has in-degree=out-degree and the underlying undirected graph has no cut vertex.

This implies that there exists a pair of adjacent vertices u and v such that the underlying undirected graph of $G - \{u, v\}$ is connected (for a proof see [195]). Apply induction on $G - \{u, v\}$ and pick all outgoing arcs from u and v. This implies that

$$\begin{aligned} a(G) &\geq a(G-u-v) + \frac{d_G(u)+d_G(v)}{2} \\ &\geq \frac{m-(d_G(u)+d_G(v)-1)}{2} + \frac{1}{2}\lceil (n-3)/2 \rceil + \frac{d_G(u)+d_G(v)}{2} \\ &= \frac{m}{2} + \frac{1}{2}\lceil (n-3)/2 \rceil + \frac{1}{2} \\ &\geq \frac{m}{2} + \frac{1}{2}\lceil (n-1)/2 \rceil. \end{aligned}$$

It is easy to verify that the resulting set of arcs forms an acyclic subgraph, and can be found in the claimed time bound. □

Theorem 4.15 *Let G be an oriented directed graph on n vertex and m arcs. Let c be the number of components in the underlying undirected graph. Then given an integer k, we can determine whether or not G has at least $\frac{m}{2} + k$ arcs which forms an acyclic subgraph in time $O(c2^{O(k^2)}k^2 + m + n^3)$.*

Proof: First, find all the c components of the underlying undirected graph corresponding to G. If $k \leq 1/2\lceil (n-c)/2 \rceil$, then G has an acyclic subgraph with at least $\frac{m}{2} + k$ arcs, else $k > 1/2\lceil (n-c)/2 \rceil \geq (n-c)/4 - 1$ or $n \leq 4k + 4 + c$. Thus n_i, the number of vertices in the i-th component is at most $n - (c-1) \leq 4k + 5$. Hence the number of arcs in each of the components is $O(k^2)$. By trying all subsets of the arcs in the component, we can find m_i, the maximum number of arcs of the i-th component that form an acyclic subgraph, for any i, in $O(2^{O(k^2)}k^2)$ time. If $\sum_{i=1}^{c} m_i \geq \lceil m/2 \rceil + k$ then G has an acyclic subgraph with at least $\frac{m}{2} + k$ arcs, else G does not have an acyclic subgraph with at least $\frac{m}{2} + k$ arcs. □

One can also ask the question of whether the given directed graph has a set of at least $\lceil \frac{m}{2} \rceil + \lceil \frac{n-1}{4} \rceil + k$ arcs that form an acyclic subgraph. The parameterized complexity of this problem remains open.

4.6 An Improved Algorithm for MAXDAG and its variant

In this Section we first develop a non trivial exact algorithm for the optimization version of the MAXDAG problem. Observe that an algorithm with time complexity $O(2^m n^{O(1)})$ is easy, where m is the number of arcs in the input directed graph. More precisely, given a directed graph $G = (V, E)$ on n vertices, we give an exact algorithm with running time $O(2^n n^{O(1)})$, to find a maximum sized subset of arcs $D \subseteq E$ such that $G' = (V, D)$ is a directed acyclic graph. Then we give improved parameterized algorithms for MAXDAG and its variants as applications of the exact algorithm developed for its optimization version. To do so we first develop a kernel of size at most $4k/3$ for the number of vertices for MAXDAG. Then applying the $O(2^n n^{O(1)})$ time algorithm for the optimization version on this kernel, we obtain an $O(2^{4k/3} n^{O(1)})$ time algorithm. We further improve this to $O(2^k n^{O(1)})$ time algorithm using a better branching technique. Then we give an improved algorithm for *above guarantee MAXDAG*.

4.6.1 Optimization Version

Given an ordering π of vertices of G, if there is an arc (i, j) such that $\pi(i) < \pi(j)$ then we call it a *forward arc*, else we call it a *backward arc*. Given a directed graph $G = (V, E)$, FAS can also be viewed as a linear ordering of the vertices such that the number of backward arcs is minimized. Or equivalently:

> DIRECTED FEEDBACK ARC SET (FAS): Given a directed graph $G(V, E)$, find a permutation $\pi : V \rightarrow \{1, 2, \cdots, |V|\}$ such that
>
> $$\sum_{(e=(u,v)\in E,\ \pi(u)>\pi(v))} \mathbf{1}$$
>
> is minimized.

Given a permutation π, let $s(\pi)$ denote the number of backward arcs with respect to the permutation π. Our dynamic programming algorithm is based on the fact that a minimum feedback arc set possesses an optimal substructure. Let $X(S)$

denote the number of backward arcs in an optimal ordering minimizing backward arcs on the graph $G[S]$ where $S \subseteq V$. Then $X(S)$ is recursively defined as follows:

$$X(S) = \min_{u \in S} \left\{ X(S-u) + \sum_{((u,v) \in E \ \& \ v \in (S-u))} \mathbf{1} \right\}. \tag{4.2}$$

The correctness of the above recurrence is easy to verify. From now on by the phrase *optimal permutation* we mean a permutation π such that the number of backward arcs is minimized.

Now we give our algorithm FASD, see Figure 4.3, to find a minimum sized directed feedback arc set.

```
Algorithm FASD($G$)
Input: A Directed graph $G = (V, E)$.
Output: Size of a Minimum Feedback Arc Set of $G$.

Step 1: Let $Y$ be a $2^n \times 2$ multidimensional array indexed from 0
        to $2^n - 1$, initialized to $Y[S,1] = \infty$, $Y[S,2] = \emptyset$ for all subsets
        $S \subseteq V$ and $S \neq \emptyset$. $Y[\emptyset,1] = Y[\emptyset,2] = 0$.

Step 2: for $S \subseteq V$ enumerated in increasing order of cardinality do

        Step 3: For every vertex $u \in V - S$:

                Let, $P = Y[S,1] + \sum_{((u,v) \in E \ \& \ v \in (S-u))} \mathbf{1}$.

            Step 4a: If $P = Y[S \cup \{u\}, 1]$ then
                1. $Y[S \cup \{u\}, 2] = Y[S \cup \{u\}, 2] \cup \{u\}$.
            Step 4b: If $P < Y[S \cup \{u\}, 1]$ then
                1. $Y[S \cup \{u\}, 1] = P$.
                2. $Y[S \cup \{u\}, 2] = u$.

Step 5: return $Y[V, 1]$.
```

Figure 4.3: Exact Algorithm for Finding a Minimum Size Feedback Arc Set in a Directed Graph

We have a multi-dimensional array Y of size $2^n \times 2$ which for every subset $S \subseteq V$ stores the value $X(S)$ of an optimum permutation and a set of vertices which are the possible last vertices of optimal permutations for the induced graph $G[S]$. More precisely, given a subset $S \subseteq V$, we have:

- $Y[S, 1] = X(S)$,
- $Y[S, 2] = \{v \mid \ v \in V$ such that $X(S)$ is minimized in Equation (4.2) $\}$ or a set of vertices which are possible last vertices in an optimal permutation for $G[S]$.

The correctness of the algorithm presented in Figure 4.3 follows from Equation (4.2). To see the time complexity of the algorithm observe that for every subset $S \subseteq V$ the algorithm takes $O(n)$ time. This gives the following theorem:

Theorem 4.16 *Let $G = (V, E)$ be a directed graph with n vertices and m arcs. Then the size of a minimum feedback arc set in G can be found in $O^*(2^n)$ time and $O^*(2^n)$ space.*

In fact we can also count all optimal permutations in the same time as finding the size of an optimal permutation by keeping an extra entry $Y[S, 3]$ for every $S \subseteq V$ and making some simple modifications in algorithm FASD. $Y[S, 3]$ stores the number of optimal permutation for $G[S]$. Initialize $Y[S, 3] = 0$ for all $S \subseteq V$ and $S \neq \emptyset$ and $Y[\emptyset, 3] = 1$. Whenever $P = Y[S \cup \{u\}, 1]$ in Step $4a$ of FASD then do $Y[S \cup \{u\}, 3] = Y[S \cup \{u\}, 3] + Y[S, 3]$ and if $P < Y[S \cup \{u\}, 1]$ in Step $4b$ of FASD then do $Y[S \cup \{u\}, 3] = Y[S, 3]$. The value in $Y[V, 3]$ gives us the total number of optimal permutations for G.

Theorem 4.17 *Let $G = (V, E)$ be a directed graph with n vertices and m arcs. Then we can count the number of minimum sized feedback arc sets in G in $O^*(2^n)$ time and $O^*(2^n)$ space.*

Observe that if we actually want an optimal permutation then we can obtain this by following the values stored at $Y[S, 2]$. We start from $Y[V, 2]$, which stores a set of vertices which are possible last vertices of an optimal permutation for G, and trace back following the list stored in $Y[S, 2]$. Suppose we want to enumerate all optimal permutations. We can do this recursively by trying every vertex v

in $Y[V, 2]$ as the last vertex of an optimal permutation and then looking for an optimal permutations of $G[V - \{v\}]$. Observe that after we have filled the array in the algorithm $\mathsf{FASD}(G)$, we can enumerate all optimal permutations of G in polynomial delay.

Theorem 4.18 *Let $G = (V, E)$ be a directed graph with n vertices and m arcs. Then all the permutations π of V corresponding to minimum size feedback arc sets in G can be enumerated in $O^*(2^n + Z)$ time where Z is the total number of such permutations for G. After initial $O^*(2^n)$ time to find an optimal permutation, every other optimal permutation is enumerated after polynomial delay.*

In Theorems 4.16 and 4.17, we used an array of size $O^*(2^n)$ time to find an optimal permutation for G. We can reduce the exponential space requirement to find a minimum size feedback arc set or to count all minimum size feedback arc sets to polynomial space at the expense of increased running time. The usual trick is to apply the divide-and-conquer paradigm to reduce the space requirement. This has been used in [140, 25] for other problems.

The idea is to guess the middle vertex $v \in V$ of the optimal permutation and recurse on all possible partitions P_1, P_2 of $V - \{v\}$ such that $||P_1| - |P_2|| \leq 1$. Observe that there are at most 2^n such partitions and given a partition P_1, P_2, P_1 or P_2 could be either side of the vertex v. Hence we have at most 2^{n+1} possible legal partitions. This gives us the following recurrence for the polynomial space algorithm:

$$T(n) = 2^{n+1} n^{O(1)} T\left(\frac{n}{2}\right).$$

This recurrence solves to $O^*(4^n n^{O(\log n)})$ which gives us the following theorem:

Theorem 4.19 *Let $G = (V, E)$ be a directed graph with n vertices and m arcs. Then the size of a minimum feedback arc set and total number of minimum size feedback arc sets in G can be found in $O^*(4^{n+o(n)})$ time and polynomial space.*

We remark that Theorems 4.18, 4.19 and 4.16 can be generalized to the weighted case where every arc has been assigned a positive real weight and the objective is to find a maximum weight arc induced acyclic subgraph or to count or enumerate all the maximum weight arc induced acyclic graphs. For the weighted case we have the following theorem.

Theorem 4.20 *Let $G = (V, E)$ be a directed graph with n vertices and m arcs and let w be a weight function $w : E \to \mathbb{R}^+$. Then*

1. *a minimum weight feedback arc set can be found in $O^*(2^n)$ time and $O^*(2^n)$ space;*

2. *the total number of minimum weight feedback arc set can be counted in $O^*(2^n)$ time and $O^*(2^n)$ space;*

3. *all the minimum weight feedback arc sets can be enumerated in $O^*(2^n + Z)$ time where Z is the total number of minimum weight feedback arc sets of G. After initial $O^*(2^n)$ time to find a minimum weight feedback arc set, every other minimum weight feedback arc set is enumerated after polynomial delay.*

Now we give various applications of Theorem 4.16 in obtaining improved parameterized algorithms for variants of MAXDAG.

4.6.2 Parameterized MAXDAG Revisited

We first find a kernel for MAXDAG as a function of n rather than as a function of m. We obtained a kernel having at most $2k$ arcs in Section 4.5.2.

Kernel for MAXDAG

Given a graph $G = (V, E)$, we preprocess it by doing the following steps:

(R0) If $k \leq |E|/2$ then answer YES.

(R1) Remove vertices of in-degree or out-degree 0.

(R2) Let v be a vertex of in-degree = out-degree = 1 having u as the in-neighbor and w as its out-neighbor. Then remove v and add the arc (u, w).

By Lemma 4.9, we know that every directed graph has an acyclic subgraph of size at least $\lceil |E|/2 \rceil$ that can be obtained in polynomial time. This ensures the soundness of $(R0)$. The soundness of $(R1)$ and $(R2)$ follows from the following lemma which is easy to show.

Lemma 4.12 *Let $G = (V, E)$ be a directed graph and $G' = (V', E')$ be the directed graph obtained after applying one iteration of reduction rules $(R0)$ to $(R2)$. Then*

(1) if we apply $(R1)$ on some vertex v then G has an acyclic subgraph of size k if and only if G' has an acyclic subgraph of size $k - d'(v)$. Here $d'(v)$ is either $d^+(v)$ or $d^-(v)$ depending on whether v is a vertex of in-degree 0 or out-degree 0.

(2) if we apply $(R2)$ on some vertex v then G has an acyclic subgraph of size k if and only if G' has an acyclic subgraph of size $k - 1$.

Lemma 4.12 gives us the following kernelization lemma.

Lemma 4.13 *Let $(G = (V, E), k)$ be a "yes" instance of* MAXDAG *and $(G' = (V', E'), k')$ be the reduced instance of $(G = (V, E), k)$ after applying the rules $(R0) - (R2)$ exhaustively. Then $|V'| \leq 4k'/3$.*

Proof: Observe that when we can not apply reduction rules $(R1)$ and $(R2)$ then every vertex in G' has total degree (in-degree + out-degree) at least 3. This implies that $m \geq 3|V'|/2$ and $(R0)$ implies that $m \leq 2k'$. Combining these two inequalities we get $|V'| \leq 4k'/3$. □

An Improved Algorithm for MAXDAG

The following corollary follows from Theorem 4.16 and Lemma 4.13, improving on Theorem 4.14.

Corollary 4.3 *Let $G = (V, E)$ be a directed graph and k be a positive integer. Then we can determine whether G has an acyclic subgraph of size k or not in time $O(2^{4k/3} n^{O(1)}) = O((2.5198)^k n^{O(1)})$.*

We can further improve this to $(2^k n^{O(1)})$ time by obtaining a kernel with at most k vertices after a branching technique.

We give two more reduction rules.

(R3) Let v be a vertex of in-degree 1 and out-degree $r(> 1)$ having u as the in-neighbor and $w_1, w_2, \cdots, w_r$ as its out-neighbors. Then remove v and add arcs (u, w_i), $1 \leq i \leq r$.

(R4) Let v be a vertex of out-degree 1 and in-degree $r(>1)$ having w as the out-neighbor and $u_1, u_2, \cdots, u_r$ as its in-neighbors. Then remove v and add arcs $(u_i, w),\ 1 \leq i \leq r$.

Lemma 4.14 *Let $G = (V, E)$ be a directed graph and v be a vertex of in-degree 1 (or out-degree 1). Let G' be the graph obtained from G by applying reduction rule (R3) [(R4)]. Then G has an acyclic subgraph of size k containing $(u, v)[(v, w)]$ if and only if G' has an acyclic subgraph of $k - 1$.*

Proof: Let $D \subseteq E$ be of size at least k, containing (u, v), such that $G^* = (V, D)$ is acyclic. Let $B = \{(u, v), (v, w_1), \cdots, (v, w_l)\}$ be the set of arcs in D which contain v as one of its endpoint. Then take $D' = D - B + \{(u, w_1), \cdots, (u, w_l)\}$ as an acyclic subgraph of size at least $k - 1$ for G'.

Let D' be the subset of arcs of size at least $k - 1$ of $G' = G - \{v\}$ such that it forms an acyclic subgraph of G'. Consider the topological ordering of $G''(V - \{v\}, D')$. Now place the vertex v in a way that it comes after u in the topological order. Let $X = \{(u, w_i) \mid (u, w_i) \in D',\ w_i \text{ out-neighbor of } v\}$. Now take $D = D' - X \cup \{(v, w_i) \mid (u, w_i) \in X\} \cup \{(u, v)\}$ as an acyclic subgraph of size k containing (u, v) in G. □

We conjure all that we have developed so far and use it to give an improved algorithm for MAXDAG. The algorithm is given in the Figure 4.4.

Now we argue about the correctness and time complexity of the algorithm. The correctness of Steps 0 and 1 is clear. In Steps 2 and 3 we branch on the arc (u, v) and (v, w) respectively. In each of these Steps we further branch on two cases that is either $(u, v) \in D$ or $(u, v) \notin D$ and return the larger size solution. The correctness of Steps 2a and 3a follows from Lemma 4.14. In Steps 2b and 3b, we are looking for an acyclic subgraph without (u, v) or (v, w). This implies that (u, v) or (v, w) is part of the directed feedback arc set and hence can be deleted from G which makes v either a vertex of in-degree or out-degree 0. This implies that there exists a solution D of size k containing all out-arcs or in-arcs of v.

Observe that when we apply either Step 2 or Step 3 the total degree of every vertex v is at least 3. Hence, after we branch in Steps 2a and 3a the parameter k reduces by 1 while in Steps 2b and 3b the parameter k at least reduces by 2. This

```
Algorithm MADS($G$,$k$,$D$)
Input: A Directed graph $G = (V, E)$.
Output: A subset of arcs $D \subseteq E$ of size at least $k$ such that the
induced subgraph on $D$ is acyclic if it exists or NO otherwise .

Step 0: If $k \leq |E|/2$ then find a set of arcs of size $|E|/2$ forming an
        acyclic subgraph in polynomial time and return it as $D$.
Step 1: Obtain a graph $(G', k')$ by applying reduction rules $(R1)$
        and $(R2)$ recursively on $(G, k)$. Now $k \leftarrow k'$.
Step 2: If there exists a vertex $v$ of in-degree 1 with in-neighbor $u$
        then branch as in Steps $2a$ and $2b$ and return the solution of
        larger size.
        Step 2a: $D \leftarrow D \cup \{(u, v)\}$. Apply $(R3)$ on $G'$ and call
              MADS($G'$,$k-1$,$D$).
        Step 2b: $D \leftarrow D \cup \{(v, w_i) \mid (v, w_i) \in E\}$. Call MADS($G -
              \{v\}$, $k - d^+(v)$,$D$).
Step 3: If there exists a vertex $v$ of out-degree 1 with out-neighbor
        $w$ then branch as in Steps $3a$ and $3b$ and return the solution
        of larger size.
        Step 3a: $D \leftarrow D \cup \{(v, w)\}$. Apply $(R4)$ on $G'$ and call
              MADS($G'$,$k-1$,$D$) .
        Step 3b: $D \leftarrow D \cup \{(w_i, v) \mid (w_i, v) \in E\}$. Call MADS($G -
              \{v\}$, $k - d^-(v)$,$D$).
Step 4: If $|V'| > k$ then return NO else apply Theorem 4.16 on $G$
        and obtain a $D$ and return it.
```

Figure 4.4: Improved Parameterized Algorithm for MAXDAG

gives the following recurrence on the parameter k:

$$T(k) \leq T(k-1) + T(k-2). \tag{4.3}$$

When we reach Step 4 of the algorithm then every vertex of G has in-degree as well as out-degree at least 2 and hence $|E| \geq 2|V|$. By Step 0 we know that $k \geq |A|/2$. This gives us that

$$2|V| \leq |E| \leq 2k \Rightarrow |V| \leq k.$$

This implies that we have obtained a graph G with at most k vertices and $2k$ arcs. Now we apply Theorem 4.16 on G and solve the maximum acyclic subgraph problem in $O(2^k n^{O(1)})$ time. The recurrence 4.3 solves to $O(1.62^k n^{O(1)})$ and hence the running time of the algorithm is bounded by $O(2^k n^{O(1)})$. This gives us the following theorem:

Theorem 4.21 *Let $G = (V, E)$ be a directed graph and k be a positive integer. We can determine whether G has an acyclic subgraph of size k or not in $O(2^k n^{O(1)})$ time.*

4.6.3 Above Guarantee MAXDAG

Now we apply Theorem 4.16 to obtain an improved parameterized algorithm for the above guarantee version of MAXDAG in oriented directed graphs.

By Lemma 4.11, we know that any oriented directed graph $G = (V, E)$ with m arcs and n vertices, with the underlying undirected graph having c components, has an acyclic subgraph with at least $\frac{m}{2} + 1/2\lceil (n-c)/2 \rceil$ arcs. First, we find all the c components of the underlying undirected graph corresponding to G. If $k \leq 1/2\lceil (n-c)/2 \rceil$, then G has an acyclic subgraph with at least $\frac{m}{2} + k$ arcs, else $k > 1/2\lceil (n-c)/2 \rceil \geq (n-c)/4 - 1$ or $n \leq 4k + 4 + c$. Thus n_i, the number of vertices in the i-th component, is at most $n - (c-1) \leq 4k + 5$. So now apply Theorem 4.16 to every connected component containing at most $4k + 5$ vertices. This gives us the following theorem:

Theorem 4.22 *Let G be an oriented directed graph on n vertices and m arcs. Then given an integer k, we can determine whether or not G has at least $\frac{m}{2} + k$ arcs that form an acyclic subgraph in $O(16^k n^{O(1)})$ time.*

Theorem 4.22 improves the algorithm given in Theorem 4.15

4.6.4 Directed graphs with minimum out degree $f(n)$

In this Section we show that if the minimum out-degree or in-degree of a graph is at least $f(n)$, for any function of n, then the directed feedback arc set problem is fixed parameter tractable for such graphs. Here $f(n)$ could be a slow growing function such as $\log^* n$. Our algorithm depends on the following combinatorial lemma which relates the size of a minimum feedback arc set and the minimum out-degree or in-degree of the graph.

Lemma 4.15 *Let $G = (V, E)$ be a directed graph with minimum out-degree t (or minimum in-degree t). Then any feedback arc set of G must contain at least $\binom{t+1}{2}$ arcs.*

Proof: Without loss of generality assume that the minimum out-degree of the graph is at least t. Let F be any feedback arc set of G. Then by definition of F, $G-F$ is a directed acyclic graph and hence there exists a topological ordering of $G-F$. Let this ordering of vertices be $P = v_1, v_2, \cdots, v_n$. Let $S = \{v_{n-t+1}, \cdots, v_n\}$. Since v_n has out degree 0 in $G-F$, all arcs out of v_n in G are part of F. Similarly for v_{n-k}, $0 \leq k \leq t-1$, at least $t-k$ arcs emanating from v_{n-k} in G are part of F. Hence

$$|F| \geq \sum_{k=0}^{t-1}(t-k) = \binom{t+1}{2}.$$

This shows that any minimum feedback arc set of G must contain at least $\binom{t+1}{2}$ arcs. □

Lemma 4.15 directly gives a fixed parameter tractable algorithm for the directed feedback arc set problem in graphs with minimum out-degree or in-degree $f(n)$ by the kernelization technique. Given a graph G with minimum out-degree or in-degree $f(n)$ and a positive integer k, we check whether $k < \binom{f(n)+1}{2}$. If $k < \binom{f(n)+1}{2}$ then we return NO. Otherwise $k \geq \binom{f(n)+1}{2}$ and hence $n \leq g(k)$ for some function g which is a function of k alone. Now we apply Theorem 4.16 and solve the parameterized directed feedback arc set problem in time $O(2^{g(k)} n^{O(1)})$. This gives the following theorem.

Theorem 4.23 *Let $G = (V, E)$ be a directed graph such that the minimum out-degree or in-degree of the graph is at least $f(n)$, for some fixed function f. Then the directed feedback arc set problem is fixed parameter tractable in G.*

4.7 Conclusion

In this chapter, we have obtained efficient algorithms for the parameterized feedback arc and vertex set problems on weighted tournaments. For the feedback arc set problem, the complexity of the algorithms in unweighted and weighted (with weights at least 1) versions are the same while this is not the case for the feedback vertex set problem.

We have also given FPT algorithms for the parametric duals of the directed feedback vertex and arc set problems in oriented directed graphs and directed graphs, respectively. The dual of the directed feedback vertex set problem in directed graphs is shown to be $W[1]$-hard. Finally we gave improved algorithm for the dual of directed feedback arc set problem through new kernelization and efficient exact algorithms for their optimization versions. This approach of obtaining faster parameterized algorithms for "edge" problems seems of general relevance. Another example for which we can obtain faster parameterized algorithms using this methodology is MAX-CUT. For both of these "edge" problems (MAX-CUT and MAXDAG) we obtain an improved kernel by bounding n as a function of k which is crucial for the improved FPT algorithms. It will be interesting to find some other "edge" problems where bounding n (instead of m), as a function of k and then using the exact algorithms for the optimization version of these problems gives an improved parameterized algorithm.

5 FPT Algorithms for W-Hard Problems in Graphs with no Small Cycles

Independent Set and Dominating Set are problems that are well known to be hard problems in the parameterized complexity hierarchy [79]. In this chapter we study these problems and several of their variants and show them to be fixed parameter tractable on graphs that have no short cycles – more specifically on graphs with no triangles or 4-cycles. Essentially we show that the existence of small cycles make these problems hard in general graphs. These problems are known to be NP-complete on such graphs as well [6, 7]. We also look at the special case of the Set Cover problem where the intersection between any pair of its sets in the instance is bounded by a fixed constant. While the general version of Set Cover is known to be $W[2]$-complete, we prove this special version fixed parameter tractable.

Organization of the rest of the Chapter: In Section 5.1, we look at the Dominating Set problem and show that the problem is $W[2]$-complete even in bipartite graphs and split graphs (a graph in which the vertices can be partitioned into a clique and an independent set). Though variations of Dominating Set like Red-Blue Dominating Set [82] and Constrained Dominating Set [111] have been studied before and shown to be W[2]-complete, to the best of our knowledge the standard Dominating Set problem (which we consider here) in bipartite graphs has not been studied before. Our observation means that the dominating set problem is $W[2]$-complete in triangle free graphs. Then we show that the problem is FPT if the input graph has girth at least 5 (i. e. when there are no three or four cycles). It turns out that this result can be generalized to several variants of the Dominating Set problem on graphs with girth at least five.

In Section 5.2, we look at the Set Cover problem for which Dominating Set is a special instance. Set Cover problem is known to be $W[2]$-complete [79]. Here we show that if the set cover instance satisfies the property that the intersection of any pair of its sets is bounded by a fixed constant then the problem is fixed parameter tractable.

In Section 5.3, we look at t-Vertex Cover and t-Dominating Set problems. These are generalizations of Vertex Cover and Dominating Set problems. In the t-Vertex Cover problem, we are interested in finding a set of at most k vertices covering at least t edges and in the t-Dominating Set problem the objective is to find a set of at most k vertices that dominates at least t vertices. Both these problems have been parameterized in two different ways: by k alone and by both k and t. Both these problems are fixed parameter tractable when parameterized by both k and t. Bläser [26] gave $O(2^{O(t)}n^{O(1)})$ algorithm for both the problems using color coding technique. Guo et. al. [139] have shown that t-Vertex Cover is $W[1]$-complete when parameterized by k alone. It is easy to see that the t-Dominating Set is $W[2]$-complete by a reduction from Dominating set when parameterized by k alone. We show that both these problems are fixed parameter tractable in graphs with girth at least five, when parameterized by k alone.

In Section 5.4, we look at the Independent Set problem and several of its variants. We show that these problems are fixed parameter tractable in triangle free graphs while they are W[1]-complete in general graphs.

In contrast to our results in earlier sections, in Section 5.5, we exhibit a problem that is $W[1]$-hard in graphs with no small cycles. This is the Dense Subgraph problem [168]. Here, given a graph $G = (V, E)$ and positive integers k and l, the problem is to determine whether there exists a set of at most k vertices $C \subseteq V$ such that the induced subgraph on C has at least l edges; here k is the parameter.

In Section 5.6, we deviate and look at the approximability of the Dominating Set problem. We conclude that the Dominating Set problem is as hard to approximate in bipartite graphs as in general undirected graphs. We also give an approximation algorithm of factor $O(\log p)$ for the Dominating Set problem if the input graph has girth at least 5, where p is the size of an optimum dominating set of the input graph. This improves the previously known approximation algorithm of factor $O(\log n)$, where n is the number of vertices in the input graph.

Section 5.7 gives some concluding remarks and open problems.

We assume that all our graphs are simple and undirected. Given a graph $G = (V, E)$, n represents number of vertices, and m represents the number of edges. For a subset $V' \subseteq V$, by $G[V']$ we mean the subgraph of G induced on V'. By $N(u)$ we represent all vertices (excluding u) that are adjacent to u, and by $N[u]$, we refer to $N(u) \cup \{u\}$. Similarly, for a subset $D \subseteq V$, we define $N[D] = \cup_{v \in D} N[v]$. By the *girth* of a graph, we mean the length of the shortest cycle in the graph. We say that a graph is a G_i *graph* if the girth of the graph is at least i. A vertex is said to *dominate* all its neighbors. By ln we denote the natural logarithm function.

5.1 Dominating Set and its Variants

In this section we look at the Dominating Set problem and its variants.

> Dominating Set: Given a graph $G = (V, E)$ and an integer $k \geq 0$, determine whether there exists a set $D \subseteq V$, of size at most k, such that for every vertex $u \in V$, $N[u] \cap D \neq \emptyset$.

We say that the set D "*dominates*" the vertices of G. We first show that Dominating Set problem is W[2]-complete in bipartite graphs and split graphs. Then we give a fixed parameter tractable algorithm for the problem in graphs with girth at least 5.

5.1.1 Dominating Set in Bipartite and Split Graphs

Theorem 5.1 *The* Dominating Set *problem is* $W[2]$*-complete in bipartite graphs.*

Proof: We prove this by giving a reduction from the Dominating Set problem in general undirected graphs. Given an instance $(G = (V, E), k)$ of Dominating Set, we construct a bipartite graph $H = (V', E')$. Let z_1 and z_2 be two new vertices (not in V). Now $V' = V_1 \cup V_2$ where $V_1 = \{u_1 \mid u \in V\} \cup \{z_1\}$ and

$V_2 = \{u_2 \mid u \in V\} \cup \{z_2\}$. If there is an edge (u, v) in E then we draw the edges (u_1, v_2) and (v_1, u_2). We also draw edges of the form (u_1, u_2) for every $u \in V$. Finally, we add an edge from every vertex in V_1 to z_2. This completes the construction of H.

We show that G has a dominating set of size k if and only if H has a dominating set of size $k + 1$. Let D be a dominating set of size k in G. Then clearly $D' = \{u_1 \mid u \in D\} \cup \{z_2\}$ is a dominating set of size $k + 1$ in H. Conversely, let K be a dominating set in H of size $k + 1$. Observe that either z_1 or z_2 must be part of K as z_2 is the unique neighbor of z_1. Without loss of generality, we can assume that $z_2 \in K$, as otherwise we could delete z_1 and include z_2 in K and still have a dominating set of size at most $k+1$ in H. Now take $D = \{u \mid u \in V,\ u_1 \textit{ or } u_2 \in K\}$. Clearly D is of size k. We show that D is a dominating set in G. For any $u \notin D$, $u_2 \notin K$ and hence there exists some $v_1 \in K$ such that v_1 dominates u_2 in H. But this implies $v \in D$ and $(v, u) \in E$, which shows that v dominates u. This proves that D is a dominating set of size k for G and establishes the theorem. □

Since every bipartite graph is also triangle free, we have the following corollary.

Corollary 5.1 *The* DOMINATING SET *problem is* $W[2]$*-complete in triangle free graphs.*

Theorem 5.2 *The* DOMINATING SET *problem is* $W[2]$*-complete in split graphs.*

Proof: We again prove this by giving a reduction from the DOMINATING SET problem in general undirected graphs. Given an instance $(G = (V, E), k)$ of DOMINATING SET, we construct a split graph $H = (V', E')$. We create two copies of V namely $V_1 = \{u_1 \mid u \in V\}$ and $V_2 = \{u_2 \mid u \in V\}$. If there is an edge (u, v) in E then we draw the edges (u_1, v_2) and (v_1, u_2). We also draw edges of the form (u_1, u_2) for every $u \in V$. Now we make $H[V_1]$ a complete graph by adding all arcs of the form (u_1, v_1) for every pair of vertex $u_1,\ v_1 \in V_1$. This completes the construction of H. It is easy to see that H is a split graph with $H[V_1]$ as a clique and $H[V_2]$ as an independent set.

As in the proof of Theorem 5.1, it is easy to see that G has a dominating set of size k if and only if H has a dominating set of size k. □

An undirected graph is chordal if every cycle of length greater than three possesses a chord, that is, an edge joining two nonconsecutive vertices of the cycle. It is well known that every split graph is also a chordal graph and hence Theorem 5.2 implies that the DOMINATING SET is $W[2]$-complete in chordal graphs. As a corollary of Theorem 5.2 we get the following.

Corollary 5.2 *The* DOMINATING SET *problem is* $W[2]$*-complete in chordal graphs.*

5.1.2 An FPT Algorithm for Dominating Set in G_5 Graphs

We give a fixed parameter tractable algorithm for the DOMINATING SET problem in graphs with girth at least 5 (G_5 graphs) and also observe that various other W-hard problems become tractable for G_5 graphs.

Our algorithm follows a branching strategy where at every iteration we find a vertex that needs to be included in the Dominating Set which we are trying to construct. Once a vertex is included, we can at best delete that vertex. Though the neighbors of the vertex are dominated, we can not remove these vertices from further consideration as they can be useful to dominate other vertices.

Hence we resort to a coloring scheme for the vertices, similar to the one suggested by Alber et al. in [3, 5]. At any point of time of the algorithm, the vertices are colored as below:

1. Red - The vertex is included in the dominating set D which we are trying to construct.
2. White - The vertex is not in the set D, but it is dominated by some vertex in D.
3. Black - The vertex is not dominated by any vertex of D.

Now we define the dominating set problem on the graph with vertices colored with White, Black or Red as above. We call a graph colored red, white and black as above, as a rwb-graph.

RWB-DOMINATING SET: Let G be a G_5 graph (graph with girth at least 5) with vertices colored with Red, White or Black satisfying the following conditions, and let k be a positive integer parameter. Let R, W and B be the set of vertices colored red, white and black respectively.

1. Every white vertex is a neighbor of a red vertex.
2. Black vertices have no red neighbors.
3. $|R| \leq k$

Does G have at most $k - |R|$ vertices that dominate all the black vertices?

It is easy to verify that if we start with a general G_5 graph with all vertices colored black, and color all vertices we want to include in the dominating set as red, and their neighbors as white, the graph we obtain at every intermediate step is a rwb-graph, and the problem we will have at the intermediate steps is the RWB-DOMINATING SET problem.

The following lemma essentially shows that if the rwb-graph has a black or white vertex dominating more than k black vertices, then such a vertex must be part of every solution of size at most k, if one exists.

Lemma 5.1 *Let* $(G = (R \cup W \cup B, E), k)$ *be an instance of the* RWB-DOMINATING SET *problem where* G *is a* G_5 *graph and* k *a positive integer. Let* v *be a black or white vertex with more than* $k - |R|$ *black neighbors. Then if* G *has a set of size at most* $k - |R|$ *that dominates all black vertices, then* v *must be part of every such set.*

Proof: Let D be a set of size $k - |R|$ that dominates all black vertices in G, and suppose $v \notin D$. Let X be the set of black neighbors of v which are not in D and Y be the set of black neighbors of v in D. So $|X| + |Y| > k - |R|$. Observe that for every $v_x \in X$ we have a neighbor $u_x \in D$ which is not in Y (otherwise v, v_x, u_x is a 3 length cycle). Similarly, for $x,\ y \in X,\ \ x \neq y \Rightarrow u_x \neq u_y$. Otherwise v, x, u_x, y will form a cycle of length 4. This means that $|D| \geq |X| + |Y| > k - |R|$ which is a contradiction. □

Given a rwb-graph, Lemma 5.1 suggests the following simple reduction rule.

(R1) If there is a white or a black vertex v having more than $k - |R|$ black neighbors, then color v red and color its neighbors white.

Our goal now is to pick enough white or black vertices to dominate the black vertices. So the following reduction rules are obviously justified.

(R2) If a white vertex is not adjacent to a black vertex, delete the white vertex.

(R3) If there is an edge between two white vertices, delete the edge.

(R4) If $|R| > k$, then stop and return NO.

The following lemma follows from Lemma 5.1.

Lemma 5.2 *Let $G = (R \cup W \cup B, E)$ be an instance of* RWB-DOMINATING SET *and let $G' = (R' \cup W' \cup B', E')$ be the reduced instance after applying rules* ($R1$) *to* ($R4$) *once. Let k be an integer parameter. Then G is a yes instance if and only if G' is a yes instance. That is, G has a set of size at most $k - |R|$ dominating all vertices in B if and only G' has a set of size at most $k - |R'|$ dominating all vertices in B'.*

Let G be an instance of RWB-DOMINATING SET and let G' be the reduced instance after applying the reduction rules ($R1$) – ($R4$) until no longer possible. Then we show that if G' is a yes instance (and hence G is a yes instance), the number of vertices in G' is bounded by polynomial in k. More precisely we show the following lemma.

Lemma 5.3 *Let (G, k) be a yes instance of* RWB-DOMINATING SET *and (G', k') be the reduced instance of (G, k) after applying the rules* $(R1) - R(4)$ *until no longer possible. Then, the number of vertices in G' is $O(k^3)$, that is, a kernel of size at most $O(k^3)$ can be obtained for* RWB-DOMINATING SET*.*

Proof: Let R', B' and W' be the set of vertices colored red, black and white respectively in G'. We argue that each of $|R'|$, $|B'|$ and $|W'|$ is bounded by $O(k^3)$.

Because of ($R4$) (and the fact that G' is a yes instance), $|R'| \le k$.

Because of ($R1$), every vertex colored white or black has at most $k - |R'|$ black neighbors. Also we know that no red vertex has a black neighbor. Since G' is a yes instance, there are at most k ($k - |R'|$ to be more precise) black or white vertices dominating all black vertices. Since each of them can dominate at most k black vertices, we conclude that $|B'|$ can be at most k^2.

We argue that $|W'| \le k^3$. Towards this end, we just show that every black vertex has at most k white neighbors. Since $|B'| \le k^2$, and every white vertex is adjacent to some black neighbor (because of ($R2$) and ($R3$)), the conclusion will follow.

Note that every white vertex has a red neighbor. Observe that the white neighbors of any black vertex (any vertex for that matter) will have all distinct red neighbors. I.e. if w_1 and w_2 are white neighbors of a black vertex b, then there is no overlap between the red neighbors of w_1 and the red neighbors of w_2. This is because if w_1 and w_2 have a common red neighbor r, then we will have a 4-cycle b, w_1, r, w_2, b. Since $|R'| \leq k$, it follows that a black vertex can have at most k white neighbors.

This proves the required claim. □

Thus we have the following theorem.

Theorem 5.3 *The* RWB-DOMINATING SET *problem can be solved in* $O(k^{k+O(1)} + n^{O(1)})$ *time for* G_5 *graphs.*

Proof: It is easy to see that the reduction rules ($R1$) to ($R4$) take polynomial time to execute. When none of these rules can be executed, by Lemma 5.3, we have that the number of vertices in the resulting graph is $O(k^3)$, and each vertex has at most k black neighbors. We can just try all possible subsets of size at most k of the vertex set of the reduced graph, to see whether that subset dominates all the black vertices. If any of them does, then we say YES and NO otherwise. This will take $O(k^{3k+O(1)})$ time.

Alternatively, we can apply a branching technique on the black vertices, by selecting a black vertex or any of its neighbors in the dominating set. More precisely, let v be a black vertex. Then we branch on $N[v]$ by including $w \in N[v]$ in the possible dominating set D we are constructing and look for a solution of size $k-1$ in $G - \{w\}$ where w is colored red and all its neighbors are colored white for every $w \in N[v]$. If any of the $|N[v]|$ instances return a subset of size at most k dominating all black vertices we answer YES and NO otherwise. This step leads to $|N[v]|$-way branching or $(k+1)$-way branching as the maximum degree of the reduced graph is bounded by k. This will result in an $O((k+1)! \cdot k^{O(1)}) = O((k+1)^{k+O(1)})$ time algorithm. □

Now to solve the general Dominating Set problem in G_5 graphs, simply color all vertices black and solve the resulting RWB-DOMINATING SET problem using Theorem 5.3. Thus we have

Theorem 5.4 *Parameterized* DOMINATING SET *problem can be solved in* $O(k^{k+O(1)} + n^{O(1)})$ *time for* G_5 *graphs.*

Parameterized version of CONNECTED DOMINATING SET (where one is interested in dominating set which is connected) or INDEPENDENT DOMINATING SET (where one is interested in dominating set which is independent) are also known to be W[2]-complete [79]. Since the reduction rules ($R1$)-($R4$) apply for any dominating set, using Lemma 5.3 we can obtain a kernel of size at most $O(k^3)$ for both these problems. For the INDEPENDENT DOMINATING SET problem we also check that R remains an independent set when we add a vertex to it while applying reduction rule ($R1$), else we return NO. Furthermore in the proof of the Theorem 5.3, we try all possible subsets of size at most k and look for a connected or independent dominating set, as required. This results in the following corollary.

Corollary 5.3 *Parameterized* CONNECTED DOMINATING SET *and* INDEPENDENT DOMINATING SET *problems can be solved in* $O(k^{3k+O(1)}+n^{O(1)})$ *time for* G_5 *graphs.*

A number of other variants of dominating set problem which are W[2]-hard can be shown to be fixed parameter tractable in a similar way for G_5 graphs though not using kernelization. We give necessary details for a few of them in the next subsections.

5.1.3 Red-Blue Dominating Set and Constraint Bipartite Dominating Set

We first give an algorithm for RED-BLUE DOMINATING SET problem which is defined as follows.

> RED-BLUE DOMINATING SET [82]: Given a bipartite graph $G = (V, E)$ with V bipartitioned as $V_{red} \cup V_{blue}$ and a positive integer k. Does there exist a subset $D \subseteq V_{red}$ with $|D| \leq k$ and $V_{blue} \subseteq N(D)$.

This problem is W[2]-complete [82] in general graphs. We prove

Theorem 5.5 RED-BLUE DOMINATING SET *is FPT for* G_5 *graphs.*

Proof: Any two vertices in V_{red} can have at most one common neighbor in V_{blue} as otherwise there will be a four cycle in G. Hence, the following reduction rule is justified.

(R1′) if $x \in V_{red}$ has degree more than k then include $x \in D$.

The correctness of $(R1')$ follows from the fact that if we do not select x in D then we need more than k vertices from V_{red} to dominate $N(x)$ as any vertex $y \in V_{red}$, $y \neq x$, can dominate at most one vertex of $N(x)$. After exhaustively applying reduction rule $(R1')$ if the size of D is more than k we answer NO.

Remove $N[D]$ from G, i. e., set $V_{red} = V_{red} \setminus D$ and $V_{blue} = V_{blue} \setminus N(D)$. Now the degree of every vertex in V_{red} is at most k and we are looking for a set of size at most $k - |D|$ in V_{red} such that it dominates all the vertices of V_{blue}. Since every vertex in V_{red} has degree at most k, the size of the set V_{blue} is bounded above by k^2 ($(k-|S|)k$ to be precise) else the answer is NO. We can not bound the size of the set V_{red} anymore, as we do not have any bound on the degree of the vertices in V_{blue}. So to find the desired dominating set in V_{red} (dominating all the vertices in V_{blue}) we do as follows:

- For all partitions $\mathscr{P}$ of V_{blue} into at most $k-|D|$ parts, say $\mathscr{P} = \{P_1, P_2, \cdots, P_j\}$, $1 \le j \le k-|D|$, for each P_i, $1 \le i \le j$ check whether there exists a vertex $u_i \in V_{red}$ such that $P_i \subseteq N(u_i)$. Call the partition $\mathscr{P}$ *valid* if for all $1 \le i \le j$, there exists $u_i \in V_{blue}$ such that $P_i \subseteq N(u_i)$ and the set $\{u_i \mid 1 \le i \le j\}$ is called the *witness set.* If any partition $\mathscr{P}$ is valid then return YES with the corresponding witness set else return NO.

It is easy to see that there exits a subset of V_{red} of size at most $k-|D|$ dominating all vertices of V_{blue} if and only if there exists a valid partition. Number of ways in which n indistinguishable objects can be partitioned into r ways is $\binom{n+r-1}{r-1}$ [155]. Hence the total number of partitions $\mathscr{P}$ considered for our case is upper bounded by

$$\sum_{i=1}^{k-|D|} \binom{k^2+i-1}{i-1}.$$

This number is upper bounded by $O(k^{2k+O(1)})$, and hence the result that RED-BLUE DOMINATING SET is FPT for G_5 graphs follows. □

Next we study Constraint Bipartite Dominating Set problem which is defined as follows.

> Constraint Bipartite Dominating Set (CBDS) [111]: Given a bipartite graph $G = (V, E)$ with V partitioned as $V_1 \cup V_2$ and positive integers k_1 and k_2. Does there exist subsets $D_1 \subseteq V_1$ and $D_2 \subseteq V_2$ with $|D_1| \leq k_1$ and $|D_2| \leq k_2$ such that $V_2 \subseteq N(D_1)$ and $V_1 \subseteq N(D_2)$.

This problem is W[2]-complete in general bipartite graphs [111] and we show

Theorem 5.6 *Parameterized* Constraint Bipartite Dominating Set *is FPT for G_5 graphs.*

Proof: To solve this problem we just need to solve two instances of Red-Blue Dominating Set problem. The instances of Red-Blue Dominating Set problem we solve are:

1. $V_{red} = V_1$, $V_{blue} = V_2$ and parameter is k_1; and
2. $V_{red} = V_2$, $V_{blue} = V_1$ and parameter is k_2.

We return YES for CBDS problem if both the instances return YES and as D_1 the red-blue dominating set returned by instance 1 and as D_2 the red-blue dominating set returned by instance 2. If either of the instances of Red-Blue Dominating Set problem returns NO, then we return NO for the CBDS problem. □

5.1.4 Threshold Dominating Set

This problem generalizes Dominating Set and is formally defined as follows.

> Threshold Dominating Set (TDS) [78]: Given a graph $G = (V, E)$ and positive integers k and r. Is there a set of at most k vertices $V' \subseteq V$ such that for every vertex $u \in V$, $N[u]$ contains at least r elements of V'?

Theorem 5.7 Threshold Dominating Set *parameterized by k is FPT for G_5 graphs.*

Proof: First we observe that if $k < r$, then the answer is NO. We assume that $r \leq \log n$, as otherwise $k \geq \log n$ and we have a kernel of size at most 2^k. Now we can solve the problem by checking all subsets of size at most k for the desired threshold dominating set.

Our algorithm is again based on the following simple reduction rule whose correctness follows from Lemma 5.1.

(R1″) if $x \in V$ has degree more than k then include $x \in V'$.

So basically we select all the vertices of degree more than k of V in V' and hence if the size of V' is more than k then we answer NO.

Next we assign a color to all the vertices. We assign white color to all the vertices (including vertices in V') which have enough (at least r) neighbors in V' and black to the rest. Note that even vertices in V' can be colored black since they may have less than r neighbors in V'. Let B and W represent the set of black and white vertices respectively and set $B' = B \setminus V'$ and $W' = W \setminus V'$. Apply reduction rules $(R2)$ and $(R3)$ of Lemma 5.2 exhaustively. The rule $(R3)$ makes $G[W]$ an independent set. Now the problem reduces to finding a set S' of size at most $k - |V'|$ in $V \setminus V'$ such that $V' \cup S'$ is a desired threshold dominating set for G, in particular for the vertices of B. Since every vertex in $V \setminus V'$ has degree at most k and we are looking for S' of size at most k in $V \setminus V'$, the size of $|B|$ is bounded above by k^2, as otherwise we answer NO.

Now what we have is a generalized version of Threshold Dominating Set problem where we have a set of $j \leq k^2$ black vertices $B = \{u_1, \cdots, u_j\}$, each with a positive integer r_i ($r_i = r - |N[v_i] \cap V'|$), $1 \leq i \leq j$. We are looking for a set $S' \subseteq (W' \cup B')$ of size at most $k - |V'|$ such that for every $u_i \in B$, $|N(u_i) \cap S'| \geq r_i$ in G' where the vertex set of G' is $V(G') = B \cup W'$ and the edge set of G' is $E(G') = \{(u, v) \in E \mid u \in W',\ v \in B \text{ or } u \in B, v \in B\}$.

To solve this generalized version of Threshold Dominating Set problem, we need to generalize our partition arguments used in the Theorem 5.5 suitably. The major differences are that G' is no more bipartite and that there are vertices which need more than 1 (possibly r) vertices in the desired threshold dominating set. To overcome this difficulty, we make a multiset M from B by having r_i copies for each vertex $u_i \in B$. Clearly the size of $|M|$ is bounded above by rk^2. Now if we apply the partition idea of Theorem 5.5 it is possible that the same vertex may dominate multiple copies of the same vertex. To deal with this call a partition $\mathscr{P} = \{P_1, P_2, \cdots, P_\alpha\}$ *valid* if (a) there exists a subset $S' \subseteq B' \cup W'$ forming a *system of distinct representatives*; that is for all $1 \leq i \leq \alpha$, there exists a *distinct* $u_i \in S'$ such that $P_i \subseteq N(u_i)$ and (b) each P_i contains at most one copy of any vertex of B. The set S' is a *witness set.* So to find the desired threshold dominating set in $B' \cup W'$ we proceed as follows.

- For all partitions $\mathscr{P}$ of M in at most $k-|V'|$ parts, say $\mathscr{P} = \{P_1, P_2, \cdots, P_\alpha\}$, $1 \leq \alpha \leq k - |V'|$, we check whether $\mathscr{P}$ is a valid partition. If any partition $\mathscr{P}$ is valid then return YES with the corresponding witness set else return NO.

For a fixed partition $\mathscr{P} = \{P_1, P_2, \cdots, P_\alpha\}$, we can do the validity testing and find a corresponding witness set in polynomial time as follows. Testing for duplicate copies in P_i's are easy. For the other part we first define the set

$$I_i = \{u \in (B' \cup W') \mid P_i \subseteq N_{G'}(u)\},$$

where $N_{G'}(u)$ denotes the neighbors of u in G'. Now we make the bipartite incidence graph for the sets $\{I_1, \cdots, I_\alpha\}$, that is a bipartite graph $G^* = (X \cup Y, E'')$, where X has a vertex x_i for every set I_i and $Y = \cup_{l=1}^{\alpha} I_l$ and there is an edge between (x_i, u) if $u \in I_i$. Now finding a "valid" system of distinct representatives reduces to finding a maximum bipartite matching in G^* saturating X, which can be done in polynomial time [87].

The total number of partitions $\mathscr{P}$ considered for our case is upper bounded by

$$\sum_{i=1}^{k-|V'|} \binom{rk^2 + i - 1}{i - 1},$$

which is at most $O((rk^2)^{k+O(1)})$. Since $r \leq \log n$ and $(\log n)^k \leq n + (2k \log k)^k$ for all n and $k \leq n$, we have the desired result that THRESHOLD DOMINATING SET problem is FPT for G_5 graphs. $\square$

5.2 Set Cover with Bounded Intersection among Sets

Dominating Set problem is well known to be a special instance of the Set Cover problem defined below.

> Set Cover: Given a base set (or universe) $\mathcal{U} = \{s_1, s_2, \cdots, s_n\}$, a collection $\mathcal{S} = \{S_1, S_2, \cdots, S_m\}$ of subsets of $\mathcal{U}$ ($S_i \subseteq \mathcal{U}, 1 \leq i \leq m$) such that $\cup_{i=1}^{m} S_i = \mathcal{U}$ and a positive integers k, does there exist a sub-collection $\mathcal{S}'$ of $\mathcal{S}$ of size at most k such that $\cup_{S_j \in \mathcal{S}'} S_j = \mathcal{U}$.

Given an instance $(G = (V, E), k)$ of the Dominating Set problem, we can formulate it as an instance of the Set Cover problem by taking $\mathcal{U} = V$ and $\mathcal{S} = \{S_v = N[v] \mid v \in V\}$. It is easy to verify that G has a dominating set of size k if and only if $(\mathcal{U}, \mathcal{S})$ has a set cover of size at most k. Hence the parameterized version of the Set Cover problem is $W[2]$-complete [79].

Here, we show that a special case of the Set Cover problem, that generalizes the Dominating Set problem for G_5 graphs to be fixed parameter tractable. More specifically, we show if the Set Cover instance $(\mathcal{U}, \mathcal{S})$ satisfies the property that for any pair of sets S_i and S_j in $\mathcal{S}$, $|S_i \cap S_j| \leq c$, for a fixed constant c, then the problem is fixed parameter tractable. We call this variant of the Set Cover problem, where every pair of sets in the given family intersect in at most c elements, as Bounded Intersection Set Cover (BISC) problem.

Observe that if the input graph G of the dominating set problem is a G_5 graph then the sets in its corresponding set cover instance satisfies a property that for any pair of sets S_u and S_v in $\mathcal{S}$, $|S_u \cap S_v| \leq 2$.

Theorem 5.8 *The* BISC *problem is fixed parameter tractable.*

Proof: If there is a set $S_i \in \mathcal{S}$ such that $|S_i| > ck$ then S_i must be in every k-sized set cover. Otherwise, we need more than k sets to cover all the elements of $\mathcal{U}$ since every other set can cover at most c elements of S_i. So this gives us a following simple reduction rule:

Rule 1: Given a set cover instance, $(\mathcal{U}, \mathcal{S}, k)$, if there exists $S_i \in \mathcal{S}$ such that $|S_i| > ck$ then obtain a new reduced instance of set cover as following:

- $\mathcal{U} \leftarrow \mathcal{U} - S_i$.
- $\mathcal{S} \leftarrow \{S - S_i \mid S \in \mathcal{S}\}$. If there are multiple copies of some set, then remove all but one copy of the same.
- $k \leftarrow k - 1$

If k becomes 0 and $\mathcal{U}$ is non-empty then this is a no instance for the problem and we stop. We apply the **Rule 1** until all the sets in $\mathcal{S}$ is of size at most ck', where $k' \leq k$. As k' sets of size ck' can only cover at most $ck'^2 \leq ck^2$ elements of $\mathcal{U}$, if $|\mathcal{U}| > ck^2$ then it is a no instance of the problem. The reduction rule also ensures that every set in $\mathcal{S}$ is distinct. But then the number of distinct sets of size at most ck in $\mathcal{S}$ can be at most the number of distinct subsets of $\mathcal{U}$. This gives us that if $|\mathcal{U}| \geq 2ck$ then

$$|\mathcal{S}| = \sum_{i=1}^{ck} \binom{|\mathcal{U}|}{i} \leq ck \binom{|\mathcal{U}|}{ck} \leq ck \left(\frac{cek^2}{ck}\right)^{ck} = ce^{ck}k^{ck+1}$$

and if $|\mathcal{U}| < 2ck$ then

$$|\mathcal{S}| = \sum_{i=1}^{ck} \binom{|\mathcal{U}|}{i} \leq 2^{2ck} = 4^{ck}.$$

Now it suffices to try each sub-collection $\mathcal{S}' \subseteq \mathcal{S}$ of size k and return YES if any of them covers the set $\mathcal{U}$ and NO otherwise. This has following time complexity:

$$\binom{ce^{ck}k^{ck+1}}{k} \leq \left(\frac{ce^{ck}k^{ck+1}e}{k}\right)^k = (ce)^k(ek)^{ck^2}.$$

Since c is a fixed constant, it follows that the running time results in a fixed parameter tractable algorithm. □

5.3 t-Vertex Cover and t-Dominating Set Problems

t-Vertex Cover and t-Dominating Set problems are respectively, generalizations of classical Vertex Cover and Dominating Set problems. Here the objective is not to cover all the edges or to dominate all the vertices but to cover at least t edges or to dominate at least t vertices with at most k vertices. More precisely they are defined as follows:

t-Vertex Cover: Given a graph $G = (V, E)$ and positive integers k and t, does there exist a set of at most k vertices $C \subseteq V$ such that $|\{e = (u, v) \in E \mid C \cap \{u, v\} \neq \emptyset\}| \geq t$.

t-Dominating Set: Given a graph $G = (V, E)$ and positive integers k and t, does there exist a set of at most k vertices $D \subseteq V$ such that $|N[D]| \geq t$.

The t-Vertex Cover and t-Dominating Set problems have been parameterized in two ways. They are either parameterized by k or by t and k. Both these problems are FPT when parameterized by both k and t [26] and are hard for different level of W-hierarchy when parameterized by k alone. t-Vertex Cover is $W[1]$-complete [139] and t-Dominating Set is $W[2]$-complete when parameterized by k alone.

Here, we first give a simple algorithm for the t-Vertex Cover when parameterized by both t and k and then show that this problem is FPT even when parameterized by k alone in G_5 graphs. We then extend this result to the t-Dominating Set problem for G_5 graphs when parameterized by k alone.

Our algorithms for the t-Vertex Cover depend on the following lemma.

Lemma 5.4 *Let $(G = (V, E), k, t)$ be a yes instance of the* t-Vertex Cover *and v be a vertex of maximum degree in G. Then there exists a t-vertex cover C whose intersection with $N[v]$ is nonempty, i.e. $N[v] \cap C \neq \emptyset$.*

Proof: Since G is a yes instance of the problem there exists a t-vertex cover C of size at most k and covering at least t edges. If $N[v] \cap C = \emptyset$ then choose $C' = C - \{u\} + \{v\}$ where u is any vertex in C. Since v is a vertex of highest degree and none of its neighbors is in C, C' also covers at least t edges and is of size at most k. $\square$

Suppose that the given graph has maximum degree bounded by d. Since there exists a t-vertex cover containing either a maximum degree vertex u or one of the neighbors of u, we can branch on u and on each of the (at most) d neighbors of u giving rise to a $(d + 1)$-way branching. The following theorem is immediate from this.

Theorem 5.9 *Let G be a graph with maximum degree d. Then* t-Vertex Cover *can be solved in $O((d + 1)^k n)$ time.*

Given a graph $G = (V, E)$ and positive integer parameters t and k, if there exists a vertex of degree at least t then we get a t-vertex cover by choosing the vertex. So without loss of generality, we can assume that every vertex has degree at most $t - 1$. Then from Theorem 5.9, we have

Corollary 5.4 t-Vertex Cover *can be solved in* $O(t^k n)$ *in general graphs.*

Suppose, instead of trying to cover at least t edges, we want to cover all but t edges (where t is a parameter) using at most k vertices. That is, we want an induced subgraph on $n - k$ vertices with at most t edges. We call it the $(m - t)$-Vertex Cover problem. Such a parameterization is known as dual parameterization and dual problems are, in general, natural and equally interesting [79, 160]. For example Vertex Cover is fixed parameter tractable whereas the dual of Vertex Cover is the Independent Set problem (which is the same as choosing $n - k$ vertices to cover all edges) and is W[1] complete.

The $(m - t)$-Vertex Cover problem can also be parameterized in two ways, by k alone and by k and t. When we have both t and k as parameters then we solve this problem by branching on an edge $e = (u, v)$. Here we branch by choosing either the vertex u or the vertex v or e which means that we are looking for a solution which contains either u or v or does not cover e. So we get the following branching recurrence:

$$T(k, t) \leq 2T(k - 1, t) + T(k, t - 1).$$

This immediately gives us the following theorem.

Theorem 5.10 $(m - t)$-Vertex Cover *can be solved in* $O(3^{t+k}(n + m))$ *time. Thus* $(m - t)$ Vertex Cover *is fixed parameter tractable if parameterized by t and* k.

When $(m - t)$-Vertex Cover problem is parameterized by k alone, we can show the following theorem.

Theorem 5.11 *The* $(m - t)$-Vertex Cover *problem is* $W[1]$*-hard when parameterized by* k *alone.*

Proof: We give a reduction from W[1]-complete t-Vertex Cover problem where we need at most k vertices to cover at least t edges. Given $(G = (V, E), k, t_1)$, an instance of t-Vertex Cover problem, we map it to $(G = (V, E), k, t_2)$ where $t_2 = |E| - t_1$ as an instance of $(m - t)$-Vertex Cover problem. Now it is easy to see that $(G = (V, E), k, t_1)$ is a yes instance of t-Vertex Cover problem if and only if $(G = (V, E), k, t_2)$ is a yes instance of $(m - t)$-Vertex Cover problem. $\square$

Now we show that the t-Vertex Cover problem is FPT for G_5 graphs when parameterized by k alone. We will see that this result also applies to $(m - t)$-Vertex Cover problem when parameterized by k alone.

Theorem 5.12 *t-Vertex Cover is fixed parameter tractable for G_5 graphs when parameterized by k alone. The algorithm runs in $O((k + 1)^k(n + m))$ time.*

Proof: Without loss of generality we can assume that the maximum degree of this graph is not bounded by a function of k, otherwise the problem is fixed parameter tractable by Theorem 5.9. Let v_0 be a vertex of highest degree and let $v_1, v_2, \ldots, v_r$ be its neighbors. Further assume that

$$deg(v_1) \geq deg(v_2) \geq \cdots deg(v_k) \geq \cdots \geq deg(v_r).$$

Let $A = \{v_0, v_1, \cdots, v_k\}$. We show that if there exists any t-vertex cover then there is one which contains either v_0 or one of its k highest degree neighbors. More precisely, we prove the following claim:

Claim : There exists a t-vertex cover C such that $A \cap C \neq \emptyset$, if one exists.

The claim says that if there exists any t-vertex cover then there exists a t-vertex cover C containing at least one vertex of A. We then branch on the vertices of the set A, and look for a solution of size $k - 1$, covering t-deg(v_i) edges in $G - \{v_i\}$, where $0 \leq i \leq k$ and recursively use this claim on the respective subgraphs. Hence the claim proves that t-vertex cover is fixed parameter tractable.

Now we are left with proving the claim. We show the claim by contradiction. Assume to the contrary that no t-vertex cover intersects A. By Lemma 5.4 we know that there exists a t-vertex cover C containing one of v_0's neighbors. Let v_l be a neighbor of v_0 in C. Because of our assumption $l > k$. Suppose for some v_i, $1 \leq i \leq k$, $N(v_i) \cap C = \emptyset$. Then we can obtain a t-vertex cover $C' = C - \{v_l\} + \{v_i\}$ of size at most k and covering at least t edges as deg$(v_i) \geq$ deg(v_l). So we now assume that for each v_i, $1 \leq i \leq k$, $N(v_i) \cap C \neq \emptyset$. Let $B_i = N(v_i) \cap C$. Observe that for each i, B_i does not contain v_l otherwise that will imply v_0, v_i, v_l is a triangle. Suppose for some $i \neq j$, $u \in B_i \cap B_j$ then v_0, v_i, u, v_j is a cycle of length 4. Hence $B_i \cap B_j = \emptyset$ for all i, j such that $i \neq j$. So this implies that

$$\sum_{i=1}^{k} |B_i| \geq k.$$

So we have $B_i \neq \emptyset$, $B_i \subseteq C - \{v_l\}$ and their pairwise intersections are empty. But this implies

$$\sum_{i=1}^{k} |B_i| \leq |C - \{v_l\}| \leq k - 1$$

which contradicts that $\sum_{i=1}^{k} |B_i| \geq k$. This in turn proves the claim.

Since we branch on the vertices of A whose size is bounded by $k+1$, we get an algorithm of time complexity $O((k+1)^k n)$. □

Since the runtime in Theorem 5.12 was independent of t, we get

Theorem 5.13 $(m-t)$-Vertex Cover *can be solved in* $O((k+1)^k(n+m))$ *time for* G_5 *graphs when parameterized by* k *only.*

By arguments similar to those used in Theorem 5.12, we can show the following.

Theorem 5.14 t-Dominating Set *can be solved in* $O((k+1)^k n^{O(1)})$ *time for* G_5 *graphs when parameterized by* k *only.*

5.4 Independent Set and its Variants in G$_4$ Graphs

Independent Set problem asks for an induced subgraph on k vertices which only contains isolated vertices. More precisely:

> Independent Set : Given a graph $G = (V, E)$ and an integer $k \geq 0$, determine whether there exists a set of at most k vertices $I \subseteq V$ such that the subgraph induced by I does not contain any edges.

Independent Set problem is W[1]-complete for general graphs. We show that the Independent Set and some of its variants are fixed parameter tractable for *triangle free* graphs. We use *Ramsey theory* to get a kernel of size $O(k^2)$ for these problems.

Theorem 5.15 *Parameterized* INDEPENDENT SET *problem can be solved in* $O(kn + k^{O(k)})$ *in* G_4 *graphs (triangle free graphs).*

Proof: Given any two integers p and q, there exists a number $R(p,q)$ such that any graph on at least $R(p,q)$ vertices contains an independent set of size p or a clique of size q. $R(p,q)$, for various values of p and q are known as *Ramsey Numbers.* It is well known that $R(p,q) \leq \binom{p+q-2}{q-1}$ [155]. And if $n > R(p,q)$ then either an independent set of size p or a clique of size q can be found in $O((p+q)n)$ time by transforming the inductive arguments used in the proof of Theorem 27.3 in [155] for the upper bound of $R(p,q)$ to a constructive algorithm.

If $k \leq 2$, then we can check in linear time whether the graph has an independent set of size 2 or not. So let us assume that $k \geq 3$. If the number of vertices $n > k^2 \geq R(k,3)$ then we know that this graph has either an independent set of size k or a clique of size 3. But since the input graph is triangle free, we know that it must have an independent set of size k and can be found in $O(kn)$ time. Otherwise we know that $n \leq k^2$. In this case, we try all possible subsets of size at most k to see whether the graph has an independent set of size k or not. If any of them does, then we answer YES and answer NO otherwise. This will take $O(k^{O(k)})$ time. This completes the proof. $\square$

Theorem 5.15 can be extended to a larger class of problems where one is interested in finding a subset inducing a "hereditary property". A graph property Π is a collection of graphs. A graph property Π is non-trivial if Π has at least one graph and does not include all graphs. A non-trivial property is said to be *hereditary* if a graph G is in property Π implies that every *induced subgraph* of G is also in Π. Given any property Π, let $P(G,k,\Pi)$ be the problem defined below:

> $\mathrm{P(G,k,\Pi)}$: Given a graph $G = (V,E)$ and a positive integer k, determine whether there exists a set of k vertices $V' \subseteq V$ such that $G[V']$ is in Π.

Khot and Raman [160] studied this problem and showed the following theorem.

Theorem 5.16 *(Khot and Raman [160]) Let* Π *be a hereditary property that includes all independent sets but not all cliques (or vice versa). Then the problem* $P(G,k,\Pi)$ *is* $W[1]$ *hard.*

The proof of the following theorem is exactly as in the proof of Theorem 5.15, by considering the Ramsey numbers $R(k, c)$.

Theorem 5.17 *Let* Π *be a hereditary property that includes all independent sets. Then the problem* $P(G, k, \Pi)$ *restricted to* G_c *graphs for any fixed constant* $c \geq 3$ *is fixed parameter tractable and can be solved in* $O(kn + k^{O(k)}n^{O(1)})$ *time.*

Given a graph $G = (V, E)$ and a positive integer $k \geq 0$, ACYCLIC SUBGRAPH, BIPARTITE SUBGRAPH and PLANAR SUBGRAPH problems ask whether there exists a subset $V' \subseteq V$, such that $|V'| \geq k$ and $G[V']$ is acyclic, bipartite or planar respectively. All these problems are known to be W[1]-hard [79, 160] in general graphs. As a corollary to Theorem 5.17 we have following:

Corollary 5.5 ACYCLIC, BIPARTITE *and* PLANAR SUBGRAPH *problems are fixed parameter tractable with time complexity* $O(kn + k^{O(k)}n^{O(1)})$ *for* G_c *graphs for any fixed constant* $c \geq 3$.

Corollary 5.5 shows that ACYCLIC and PLANAR SUBGRAPH problems are fixed parameter tractable for bipartite graphs. In fact we can easily obtain much improved FPT algorithms for these problems for bipartite graphs. Observe that a bipartite graph has an independent set (and hence planar or acyclic induced subgraph) on $n/2$ vertices. So, if $k \leq n/2$ then for both these problems the answer is YES and otherwise $k > n/2$ or $n < 2k$ and hence we get a kernel of size at most $2k$ for both the ACYCLIC and PLANAR SUBGRAPH problems for bipartite graphs. Now we check all k sized subsets of the vertex set to see whether the subset induces an acyclic subgraph or planar subgraph. Since $\binom{n}{k} \leq \binom{2k}{k} \leq 2^{2k} = 4^k$, we get an $O(4^k n^{O(1)})$ time algorithm for both these problems for bipartite graphs.

Minimum feedback vertex set, which is a subset of vertices whose removal makes the graph acyclic, is a complement of the vertex set of the maximum ACYCLIC SUBGRAPH problem. Fomin et al. [115] have shown that the minimum feedback vertex set can be found in $O(1.7548^n)$ time in undirected graphs. So by applying this algorithm on the kernel (instead of trying all possible subsets), we get $O(1.7548^{2k}n^{O(1)}) = O(3.0793^k n^{O(1)})$ time algorithm for the ACYCLIC SUBGRAPH problem. Putting together everything we get the following theorem.

Theorem 5.18 *The parameterized* ACYCLIC SUBGRAPH *and* PLANAR SUBGRAPH *problems can be solved in* $O(3.08^k k^{O(1)} + n^{O(1)})$ *and* $O(4^k k^{O(1)} + n^{O(1)})$ *time, respectively, for bipartite graphs.*

Another problem which can be shown to be FPT for G_c graphs for any fixed constant $c \geq 3$ is the IRREDUNDANT SET problem, which is known to be W[1]-complete [81] in general graphs.

> IRREDUNDANT SET: Given a graph $G = (V, E)$ and a positive integer k. Is there a set $V' \subseteq V$ of cardinality at least k having the property that each vertex $u \in V'$ has a private neighbor ? A private neighbor of a vertex $u \in V'$ is a vertex $u' \in N[u]$ (possibly $u' = u$) with the property that for every vertex $v \in V' \setminus \{u\}$, $u' \notin N[v]$.

Since every independent set is also an irredundant set, the following theorem can be proved along the lines of Theorem 5.15, by considering the Ramsey Numbers $R(k, c)$.

Theorem 5.19 IRREDUNDANT SET *is FPT for* G_c *graphs for any fixed constant* $c \geq 3$.

5.5 Is everything easy on graphs with no small cycles ?

In contrast to the results presented in the previous sections, here we show a problem to be $W[1]$-hard even in bipartite graphs with girth at least 6 (G_6 graphs). Observe that in graphs with large girth the CLIQUE problem is trivial. We look at DENSE SUBGRAPH problem [168] which is a generalization of the CLIQUE problem.

> DENSE SUBGRAPH: Given a graph $G = (V, E)$ and positive integers k and l, determine whether there exists a set of at most k vertices $C \subseteq V$ such that $G[C]$ has at least l edges, i.e. the induced subgraph on C has at least l edges. (Note that l is at most $\binom{k}{2}$.)

It is easy to observe that DENSE SUBGRAPH problem is W[1]-hard when parameterized by k, by a simple reduction from CLIQUE. But we show that the DENSE SUBGRAPH problem parameterized by k is W[1]-hard even in bipartite graphs with girth at least 6.

Theorem 5.20 Dense Subgraph *is* $W[1]$*-hard for bipartite graphs with girth at least* 6 *when parameterized by* k.

Proof: We give a reduction from Clique. Let (G, k) be an instance of Clique with $k \geq 3$. We make the graph $G = (V, E)$ bipartite by subdividing every edge. Let $G' = (V', E')$ be the resulting subgraph. Here, $V' = V \cup W$ where $W = \{w_{uv} \mid (u, v) \in E\}$ and E', the set of edges, consists of (u, w_{uv}) and (v, w_{uv}) for every $w_{uv} \in W$. Take $k' = k + \binom{k}{2}$ and $l = 2\binom{k}{2}$.

Observe that G' is a bipartite graph as every cycle is of even length and the girth is at least 6 as the girth of G is at least 3. We claim that G has a clique of size k if and only if G' has a subgraph on k' vertices with at least l edges. Also note that every vertex in W has degree 2 as they represent edges in the original graph. Now suppose G has a clique of size k on vertex set $C = \{v_1, v_2, \cdots, v_k\}$. Then $C' = C \cup \{w_{uv} \mid u, v \in C\}$ is a vertex set of dense subgraph in G' having k' vertices and l edges as $G[C]$ has at least $\binom{k}{2}$ edges.

Conversely, let C' be a set of k' vertices such that $G'[C']$ has at least l edges. Let $O = V \cap C'$. Clearly $G'[C']$ is bipartite with O and $N = C' - O$ as the two parts of the vertex set, and every vertex in N has degree at most 2. Since the number of edges in $G'[C'] = l = 2\binom{k}{2}$, and since every vertex in N has degree at most 2, $|N| \geq \binom{k}{2}$ and hence $|O| \leq k$. Let $t = |O|$. We claim that $t = k$. Suppose not. Then $t \leq k - 1$. Also, since $k \geq 3$, $t \geq 1$. Let n_1 and n_2 be the degree 1 and degree 2 vertices in N respectively. Since G has no multiple edges, no pair of vertices in N with degree 2 can be adjacent to the same pair of vertices in O and hence $n_2 \leq \binom{t}{2}$. Then the number of edges in $G[C']$ is:

$$\begin{aligned}
2\binom{k}{2} \leq |E(G[C'])| &= 2n_2 + n_1 \\
&= k' - t + n_2 \\
&= k + \binom{k}{2} - t + n_2 \\
&\leq k + \binom{k}{2} - t + \binom{t}{2} \qquad (\text{ since } n_2 \leq \tbinom{t}{2}).
\end{aligned}$$

From the above it implies that

$$t + \binom{k}{2} \leq k + \binom{t}{2}. \tag{5.1}$$

If $t = 1$ then

$$k + \binom{1}{2} = k < 1 + \binom{k}{2},$$

a contradiction to inequality (5.1). So let $2 \leq t \leq k-1$. But then

$$k + \binom{t}{2} \leq k + \binom{k-1}{2} = \binom{k}{2} + 1 < \binom{k}{2} + t,$$

again a contradiction to inequality (5.1). This implies that $|O| = k$. As a result of this, $|N| = \binom{k}{2}$ and every vertex in N has degree 2. Every vertex of degree 2 in N represents an edge in $G[O]$. This shows that the vertices of O form a clique in the original graph. □

5.6 Approximation of Dominating Set

In this section we give some results concerning approximation of Dominating Set problem in bipartite and G_5 graphs. We refer to [219] for all the basic definitions regarding approximation algorithms.

We first give two results concerning hardness of approximation for Dominating Set problem in bipartite and split graphs. Our results are based on the following seminal result of Feige [97] about the hardness of approximation of Dominating Set in general undirected graphs.

Proposition 5.1 ([97]) Dominating Set *problem can not be approximated in polynomial time below* $(1\text{-}o(1))\ln n$ *in general undirected graphs unless* $NP \subset DTIME(n^{O(\log\log n)})$.

We show that under the same hypothesis as in Proposition 5.1, $\left(\frac{(1-o(1))\ln n}{2}\right)$ is a threshold below which the Dominating Set problem can not be approximated in bipartite graphs. More precisely we show the following theorem.

Theorem 5.21 Dominating Set *problem can not be approximated in polynomial time below* $\left(\frac{(1-o(1))\ln n}{2}\right)$ *in bipartite graphs unless* $NP \subset DTIME(n^{O(\log\log n)})$.

Proof: We show that if we have factor α approximation algorithm for the dominating set problem in bipartite graphs then it implies 2α factor approximation algorithm for the dominating set problem in general undirected graphs. Given a graph G, we first apply Theorem 5.1 to obtain the bipartite graph H and then apply the factor α approximation algorithm for dominating set problem in bipartite graphs to get a dominating set D for H. We obtain a dominating set D' for G from the dominating set D for H as in the proof of Theorem 5.1. Let OPT_G denote the size of an optimum dominating set for the graph G. Now note that

$$\begin{aligned}
|D'| &= |D| - 1 \\
&\leq \alpha \cdot OPT_H - 1 \\
&\leq \alpha \cdot (OPT_H - 1) + \alpha - 1 \\
&\leq \alpha \cdot OPT_G + \alpha \cdot OPT_G \qquad (\text{since } OPT_H - 1 = OPT_G) \\
&= (2\alpha) \cdot OPT_G.
\end{aligned}$$

This establishes the result. □

Similarly Proposition 5.1 together with Theorem 5.2 imply that the approximability of Dominating Set problem has the same threshold of (1-$o(1)$) $\ln n$ even for split graphs. This results in the following theorem.

Theorem 5.22 Dominating Set *problem can not be approximated in polynomial time below* (1-$o(1)$) $\ln n$ *in split graphs unless* $NP \subset DTIME(n^{O(\log\log n)})$.

An approximation algorithm of factor $O(\log n)$ is known for the Dominating Set problem using the reduction to the Set Cover problem (see discussion before Theorem 5.8 in Section 5.2) and the following proposition.

Proposition 5.2 ([153, 173]) *Let* $(\mathcal{U}, \mathcal{S})$ *be a set cover instance such that* $|U| = n$. *Then we can find a set cover* $\mathcal{S}' \subseteq \mathcal{S}$ *of size at most* $\mathcal{H}_n \cdot (OPT)$ *where* $\mathcal{H}_n = \sum_{i=1}^{n} 1/i$ *and* OPT *is the size of the optimum solution of the set cover instance.* $\mathcal{H}_n \leq \ln n + 1$.

Here we outline a slightly improved approximation algorithm for Dominating Set problem in G_5 graphs. This approximation algorithm has a factor $O(\log l)$ where l is the size of the optimum dominating set. The idea of the algorithm is to use the reduction rules developed in Section 5.1.2 and obtain an instance of size $O(l^3)$ with the property that maximum degree of the graph is bounded by l and then use the following proposition on the corresponding set cover instance of the problem.

Proposition 5.3 ([85]) *Let $(\mathcal{U}, \mathcal{S})$ be a set cover instance such that $|U| = n$ and the size of each set $S_i \in \mathcal{S}$ is bounded by q, that is $|S_i| \leq q$. Then we can find a set cover $\mathcal{S}' \subseteq \mathcal{S}$ of size at most $(\mathcal{H}_q - 1/2) \cdot (OPT)$ where $\mathcal{H}_q = \sum_{i=1}^{q} 1/i$ and OPT is the size of the optimum solution of the set cover instance.*

Observe that the reduction rules $(R1) - (R4)$ depend on k whereras here we have an optimization problem. Hence apply reduction rules for all values for k between 1 and n and if the reduced instance as viewed as the SET COVER problem instance satisfies the hypothesis of Proposition 5.3 then we obtain a dominating set for G by applying Proposition 5.3. Finally we return the dominating set of smallest size among the ones obtained for different k. Our detailed algorithm is described below. We outline our algorithm in terms of *rwb-graphs* described in Section 5.1.2.

```
Algo-Dom-SET(G=(V,E))
(Input: A $G_5$ graph. Output: A dominating set of $G$.)

Step 1: Given an undirected graph $G = (V, E)$. Make it a rwb-graph by coloring
        all vertices of $V$ black; that is $R = \emptyset, W = \emptyset$ and $B = V$. $\mathcal{I} = \emptyset$.

Step 2: for ($j = 1$ to $n$) do as follows:

    Step 2a: Apply reduction rules $(R1)-(R4)$ on $(G = (R \cup W \cup B, E), j)$ until
             no longer possible and obtain an instance $(G_j = (R^j \cup W^j \cup B^j, E^j), j - |R^j|)$.
    Step 2b: If ($|W^j| + |B^j| \leq 2j^3$) and the maximum degree of $G_j$ is at most
             $j$ then
                 $\mathcal{I} = \mathcal{I} \cup \{(G_j = (R^j \cup W^j \cup B^j, E^j), j - |R^j|)\}$

    (In this step we obtain a set of instances which could possibly lead to an
    optimum dominating set. So we have

    $\mathcal{I} = \{(G_k = (R^k \cup W^k \cup B^k, E^k), k - |R^k|) \mid |W^k| + |B^k| \leq 2k^3$
                 and maximum degree of $G_k$ is at most $k\}$)
```

Step 3: We obtain a set cover instance $(\mathcal{U}_k, \mathcal{S}_k)$ from the reduced graph G^k by taking $\mathcal{U} = B^k$ and having sets S_u for $u \in (W^k \cup B^k)$. $S_u = N(u) \cap B^k$ if $u \in W^k$ and $S_u = N[u] \cap B^k$ if $u \in B^k$. Obtain $\mathcal{P}$, the set of instances for the set cover problem, by changing every instance in $\mathcal{I}$ to the set cover instance. That is:

$$\mathcal{P} = \{(\mathcal{U}_k, \mathcal{S}_k) \;\mid\; (G_k = (R^k \cup W^k \cup B^k, E^k), k - |R^k|) \in \mathcal{I}\}.$$

Step 4: Apply Proposition 5.3 to every instance of the set cover problem in $\mathcal{P}$ and obtain the following set of solutions

$$\mathcal{SOL} = \{\mathcal{S}'_k \mid \mathcal{S}'_k \subseteq \mathcal{S}_k, (\mathcal{U}_k, \mathcal{S}_k) \in \mathcal{P}\}.$$

Let $\mathcal{V}(\mathcal{S}'_k)$ represent the set of vertices in G_k corresponding to the sets in the collection $\mathcal{S}'_k$. Obtain the following set

$$\mathcal{DOM} = \{\mathcal{V}(\mathcal{S}'_k) \cup R_k \mid \mathcal{S}'_k \in \mathcal{SOL} \text{ \& } R^k \text{ the red vertices of } G_k\}.$$

of possible dominating sets for G and return the one with the minimum size in $\mathcal{DOM}$ as a dominating set for G.

Theorem 5.23 *Let $G = (V, E)$ be a G_5 graph on n vertices. Then the algorithm* `Algo-Dom-SET` *outputs a dominating set of size at most $(\mathcal{H}_{p+1} - 1/2) \cdot p$ in polynomial time where $H_p = \sum_{i=1}^{p} 1/i$ and p is the size of the optimum solution of a dominating set of G. That is,* `Algo-Dom-SET` *is an approximation algorithm with performance ratio of* $\ln(p+2) + 1/2$ *for the dominating set problem for G_5 graphs.*

Proof: It is clear that the algorithm `Algo-Dom-SET` takes polynomial time. Proposition 5.3 ensures that the algorithm returns a dominating set for G. Now we show that the algorithm is a factor of $\mathcal{H}_{p+1} - 1/2$ approximation algorithm for the dominating set problem for G_5 graphs which will complete the proof of the theorem.

Let l be the smallest positive integer in Step 2 of the algorithm such that $(G_l = (R^l \cup W^l \cup B^l, E^l), l - |R^l|) \in \mathcal{I}$. The reduction rules ensures that $(G = (R \cup W \cup B, E), k)$ is a *no* instance for $1 \le k \le l - 1$ and hence we have $p \ge l$.

Consider the instance $(G_p = (R^p \cup W^p \cup B^p, E^p), p - |R^p|) \in \mathcal{I}$. Observe that the instance G_p has an optimum dominating set of size $p - |R^p|$ and the maximum degree of the graph is bounded by p. When we apply the factor $(\mathcal{H}_q - 1/2)$ set cover approximation algorithm in Step 4 on the instance $(\mathcal{U}_p, \mathcal{S}_p)$, where each set in $\mathcal{S}_p$ is bounded by $p+1$, we obtain $\mathcal{S}'_p \subseteq \mathcal{S}_p$ of size at most $|\mathcal{S}'_p| \leq (\mathcal{H}_{p+1} - 1/2)(p - |R^p|)$. Now the size of the dominating set $R_p \cup \mathcal{V}(\mathcal{S}'_p)$ corresponding to this instance for G is :

$$\begin{aligned} |R_p| + |\mathcal{V}(\mathcal{S}'_p)| &= |R_p| + |\mathcal{S}'_p| \\ &\leq |R_p| + (\mathcal{H}_{p+1} - 1/2)(p - |R^p|) \\ &\leq |R_p|(\mathcal{H}_{p+1} - 1/2) + (\mathcal{H}_{p+1} - 1/2)(p - |R^p|) \\ &= (\mathcal{H}_{p+1} - 1/2)p. \end{aligned}$$

Since we return a dominating set of minimum size among the sets in $\mathcal{DOM}$ as a dominating set for G it is clear that its size is also bounded by $(\mathcal{H}_{p+1} - 1/2)p$. This completes the proof. □

5.7 Conclusion and Discussions

In this chapter we showed that if the input graphs do not possess short cycles then the neighborhood problems like dominating set, independent set and their variants are fixed parameter tractable. We have also shown that the restriction on girth is optimal if we do not put further restriction on the graph classes. This is the first time, to our knowledge, the parameterized complexity of graph problems are classified by girth.

We also gave an improved approximation algorithm for DOMINATING SET problem in graphs with girth at least 5. It would be interesting to explore the possibility of improved approximation algorithms for other problems on graphs with no small cycles.

6 Directed Maximum Leaf Problem

The Maximum Leaf Spanning Tree problem (finding a spanning tree with the maximum number of leaves in a connected undirected graph) is an intensively studied problem from an algorithmic as well as a combinatorial point of view [36, 94, 71, 109, 126, 163, 213]. It fits into the broader class of spanning tree problems on which hundreds of papers have been written; see e.g. the book by Wu and Chao [228]. It is known to be NP-hard [127], and APX-hard [125], but can be approximated efficiently with a multiplicative factor 2 [213].

In this chapter, we initiate the combinatorial and algorithmic study of two natural generalizations of the problem to digraphs. We say that a subgraph T of a digraph D is an *out-tree* if T is an oriented tree with only one vertex s of in-degree zero (called *the root*). The vertices of T of out-degree zero are called *leaves.* If T is a spanning out-tree, i.e. $V(T) = V(D)$, then T is called an *out-branching* of D. Given a digraph D, the Directed Maximum Leaf Out-Branching problem is the problem of finding an out-branching in D with the maximum possible number of leaves. Denote this maximum by $\ell_s(D)$. When D has no out-branching, we write $\ell_s(D) = 0$. Similarly, the Directed Maximum Leaf Out-tree problem is the problem of finding an out-tree in D with the maximum possible number of leaves, which we denote by $\ell(D)$. Note that an out-tree with $l(D)$ number of leaves may not be spanning. Both these problems are equivalent for connected undirected graphs, as any maximum leaf tree can be extended to a maximum leaf spanning tree with the same number of leaves.

Notice that $\ell(D) \geq \ell_s(D)$ for each digraph D. Let $\mathcal{L}$ be the family of digraphs D for which either $\ell_s(D) = 0$ or $\ell_s(D) = \ell(D)$. We will show that $\mathcal{L}$ contains all

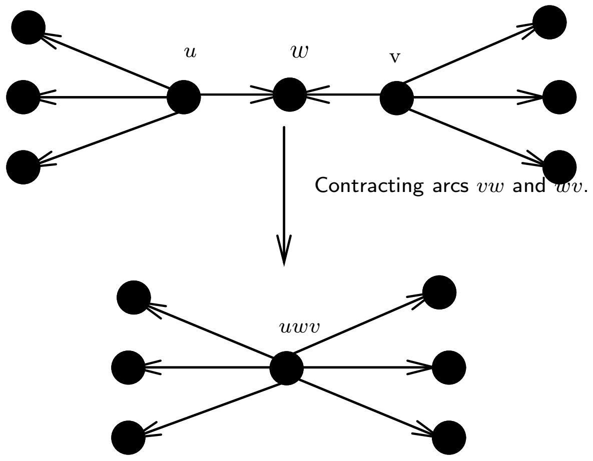

Figure 6.1: Illustrating that k-DMLOB and k-DMLOT are not minor closed.

strong and acyclic digraphs.

We investigate the above two problems from the *parameterized complexity* point of view. The parameterized version of the Directed Maximum Leaf Out-Branching (the Directed Maximum Leaf Out-tree) problem is defined as follows: Given a digraph D and a positive integral parameter k, is $\ell_s(D) \geq k$ ($\ell(D) \geq k$)? We denote the parameterized versions of the Directed Maximum Leaf Out-Branching and the Directed Maximum Leaf Out-Tree problems by k-DMLOB and k-DMLOT respectively.

While the parameterized complexity of almost all natural problems on undirected graphs is well understood, the world of digraphs is still wide open. The main reason for this anomaly is that most of the techniques developed for undirected graphs cannot be used or extended to digraphs. In what follows we briefly explain why the standard techniques for the Maximum Leaf Spanning Tree problem on undirected graphs cannot be used for its generalizations to digraphs.

- The Graph Minors Theory of Robertson and Seymour [203] is a powerful (yet non-constructive) technique for establishing membership in FPT. For example, this machinery can be used to show that the Maximum Leaf Spanning Tree problem is FPT for undirected graphs (see [108]). However, Graph Minors Theory for digraphs is still in a preliminary stage and at the moment cannot be used as a tool for tackling interesting directed graph problems.

- Bodlaender [27] used the following arguments to prove that the Maximum Leaf Spanning Tree problem is FPT: If an undirected graph G contains a star $K_{1,k}$ as a minor, then it is possible to construct a spanning tree with at least k leaves from this minor. Otherwise, there is no $K_{1,k}$ minor in G, and it is possible to prove that the treewidth of G is at most $f(k)$. Thus, dynamic programming can be used to decide whether there is a tree with k leaves. This approach does not work on directed graphs because containing a big out-tree as a minor does not imply the existence of an out-branching or out-tree with many leaves in the original graph. For an example, see Figure 6.1, here any out-tree in the original graph has at most 4 leaves but the graph obtained after contracting the arc uv has an out-tree with 6 leaves. In short, the properties of having no out-branching with at least k leaves or having no out-tree with k leaves are not minor closed.

- The most efficient approach for designing FPT algorithms for undirected graphs is based on a combination of combinatorial bounds and preprocessing rules for handling vertices of small degrees. Kleitman and West [163] and Linial and Sturtevant [171] showed that every connected undirected graph G on n vertices with minimum degree at least 3 has a spanning tree with at least $n/4 + 2$ leaves. Bonsma et al. [36] combined this combinatorial result with clever preprocessing rules to obtain the fastest known algorithm for the k-Maximum Leaf Spanning Tree problem, running in time $O(n^3 + 9.4815^k k^3)$. We don't know of such combinatorial bounds for digraphs and in this chapter we attempt to prove one.

We obtain a number of combinatorial and algorithmic results for the Directed Maximum Leaf Out-Branching and the Directed Maximum Leaf Out-tree problems. Our main combinatorial result (Theorem 6.6) is the proof that for every digraph D with $\ell_s(D) = \ell(D) > 0$ of order n with minimum in-degree at least 3, $\ell_s(D) \geq (n/4)^{1/3} - 1$. This can be viewed as a generalization of many combinatorial results for undirected graphs related to the existence of spanning trees with many leaves [135, 163, 171]. We do not know whether the last bound is tight, however we show that there are strongly connected digraphs with minimum in-degree 3 in which every out-branching has at most $O(\sqrt{n})$ leaves (Theorem 6.7). Another parallel between the worlds of directed and undirected graphs established in this chapter (and used intensively in the algorithmic part) is the relation between the number of leaves in a maximum leaf out-tree in a digraph D and the pathwidth of its underlying graph. It is easy to check (see, e.g., [24]), that every connected undirected graph of pathwidth at least k, contains a spanning tree with at least k leaves. We show (Theorem 6.9) that if a strongly connected digraph D does not contain an out-branching with k leaves, then the pathwidth of its underlying undirected graph is $O(k \log k)$.

Our main algorithmic contributions are fixed parameter tractable algorithms for the k-DMLOB and the k-DMLOT problems for digraphs in $\mathcal{L}$ ($\ell_s(D) = 0$ or $\ell_s(D) = \ell(D)$) and for all digraphs, respectively. The algorithms are based on local search and specific directed tree partition arguments. More precisely, we show that a digraph either contains a structure that can be extended to an out-branching with many leaves or the pathwidth of the underlying undirected graph is small. While local search is a widely used technique in heuristics and approximation algorithms (see, e.g., [1]) we are not aware of its applications in parameterized complexity. We find it to be of independent interest. Our contributions settle an open question of Fellows [48, 100, 142].

This chapter is organized as follows. In the next Section we give some basic definitions and known results which we make use of in later Sections. In Section 6.2.1 we start with a simple combinatorial result on the number of leaves in an out-branching. Section 6.2.2 contains the decomposition theorem based on the combinatorial result of Section 6.2.1 and its algorithmic consequences. In Section 6.3, we define the notion of locally optimum out-trees and use this in connection with tree-partitioning arguments to give improved combinatorial bounds and algorithms with better time complexity. In Section 6.4 we give improved parameterized algorithms for k-DMLOB problem in special classes of directed graphs like k-partite tournaments. In Section 6.5, we introduce a weaker version of out-branching called *pseudo-out-branching* and give a combinatorial bound on the number of leaves in a pseudo-out-branching if the minimum in-degree of the digraph is at least 2. Finally we conclude with some remarks and open problems in Section 6.6.

6.1 Preliminaries

Let D be a digraph. By $V(D)$ and $A(D)$ we represent the vertex set and arc set of D, respectively. Recall that an *oriented graph* is a digraph with no directed 2-cycle. Given a subset $V' \subseteq V(D)$ of a digraph D, let $D[V']$ denote the subgraph induced on V'. The *underlying undirected graph* $UN(D)$ of D is obtained from D by omitting all orientations of arcs and by deleting one edge from each resulting pair of parallel edges. The *connectivity components* of D are the subgraphs of D induced by the vertices of connected components of $UN(D)$. A vertex y of D is an *in-neighbor* (*out-neighbor*) of a vertex x if $yx \in A$ ($xy \in A$). The *in-degree* $d^-(x)$ (*out-degree* $d^+(x)$) of a vertex x is the number of its in-neighbors (out-neighbors). A vertex s of a digraph D is a *source* if the in-degree of s is 0.

A digraph D is *strong* if there is a directed path from every vertex of D to every other vertex of D. A *strong component* of a digraph D is a maximal strong subgraph of D. A strong component S of a digraph D is a *source strong component* if no vertex of S has an in-neighbor in $V(D) \setminus V(S)$. The following simple result gives necessary and sufficient conditions for a digraph to have an out-branching.

Proposition 6.1 ([18]) *A digraph D has an out-branching if and only if D has a unique source strong component.*

This assertion allows us to check whether $\ell_s(D) > 0$ in time $O(|V(D)|+|A(D)|)$. Thus, we will often assume, in the rest of the chapter, that the digraph D under consideration has an out-branching.

Let $P = u_1u_2 \ldots u_q$ be a directed path in a digraph D. An arc u_iu_j of D is a *forward* (*backward*) *arc for P* if $i \leq j-2$ ($j < i$, respectively). Every backward arc of the type $v_{i+1}v_i$ is called *double*.

For a natural number n, $[n]$ denotes the set $\{1, 2, \ldots, n\}$.

Now we recall the definition of tree decomposition and treewidth. A *tree decomposition* of an (undirected) graph G is a pair (X, U) where U is a tree whose vertices we will call *nodes* and $X = (\{X_i \mid i \in V(U)\})$ is a collection of subsets of $V(G)$ such that

1. $\bigcup_{i \in V(U)} X_i = V(G)$,

2. for each edge $\{v, w\} \in E(G)$, there is an $i \in V(U)$ such that $v, w \in X_i$, and

3. for each $v \in V(G)$ the set of nodes $\{i \mid v \in X_i\}$ forms a subtree of U.

The *width* of a tree decomposition $(\{X_i \mid i \in V(U)\}, U)$ equals $\max_{i \in V(U)}\{|X_i| - 1\}$. The *treewidth* of a graph G is the minimum width over all tree decompositions of G.

If in the definitions of a tree decomposition and treewidth we restrict U to be a path then we have the definitions of path decomposition and pathwidth. We use the notation $tw(G)$ and $pw(G)$ to denote the treewidth and the pathwidth of a graph G.

We also need an equivalent definition of pathwidth in terms of vertex separators with respect to a linear ordering of the vertices. Let G be a graph and let $\sigma = (v_1, v_2, \ldots, v_n)$ be an ordering of $V(G)$. For $j \in [n]$, define $V_j = \{v_i : \ i \in [j]\}$ and denote by ∂V_j all vertices of V_j that have neighbors in $V \setminus V_j$. Setting

$$vs(G, \sigma) = \max_{i \in [n]} |\partial V_i|,$$

we define the *vertex separation* of G as

$$vs(G) = \min\{vs(G, \sigma) \colon \sigma \text{ is an ordering of } V(G)\}.$$

The following assertion is well-known. It follows directly from the results of Kirousis and Papadimitriou [162] on interval width of a graph, see also [161].

Proposition 6.2 ([161, 162]) *For any graph G, $vs(G) = pw(G)$.*

6.2 Combinatorial Bounds and Algorithms - I

The following assertion shows that $\mathcal{L}$ ($\ell_s(D) = 0$ or $\ell_s(D) = \ell(D)$) includes a large number of digraphs including all strong and acyclic digraphs (and, also, well-studied classes of semicomplete multipartite digraphs and quasi-transitive digraphs, see [18] for the definitions).

Proposition 6.3 *Suppose that a digraph D satisfies the following property: for every pair R and Q of distinct strong components of D, if there is an arc from R to Q then each vertex of Q has an in-neighbor in R. Then $D \in \mathcal{L}$.*

Proof: Let T be a maximal out-tree of D with $\ell(D)$ leaves. We may assume that $\ell_s(D) > 0$ and hence D has a unique source strong component, say H. Let r be the root of T. Observe that $r \in V(H)$ as otherwise we could extend T by adding to it an arc ur, where u is some vertex outside the strong component containing r. Let C be a strong component containing a vertex from T. Observe that $V(C) \cap V(T) = V(C)$ as otherwise we could extend T by appending to it some arc uv, where $u \in V(C) \cap V(T)$ and $v \in V(C) \setminus V(T)$. Similarly, one can see that T must contain vertices from all strong components of D. Thus, $V(T) = V(D)$ and hence $l_s(D) = l(D)$ and so $D \in \mathcal{L}$. $\square$

6.2.1 Combinatorial Bound

Now we are ready to prove the main lemma of this section.

Lemma 6.1 *Let D be an oriented graph of order n with every vertex having in-degree 2 and let D have an out-branching. If D has no out-tree with k leaves, then $n \leq 2k^5$.*

Proof: Assume that D has no out-tree with k leaves. Consider an out-branching T of D with p leaves (clearly $p < k$). Start from the empty collection $\mathcal{P}$ of vertex-disjoint directed paths. Choose a directed path R from the root of T to a leaf, add R to $\mathcal{P}$ and delete $V(R)$ from T. Repeat this for each of the out-trees of $T - V(R)$. By induction on the number of leaves, it is easy to see that this process provides a collection $\mathcal{P}$ of p vertex-disjoint directed paths covering all vertices of D.

Let $P \in \mathcal{P}$ have $q \geq n/p$ vertices and let $P' \in \mathcal{P} \setminus \{P\}$. There are at most $k-1$ vertices on P with in-neighbors in P' since otherwise we could choose a set X of at least k vertices on P for which there were in-neighbors on P'. The vertices of X would be leaves of an out-tree formed by the vertices $V(P') \cup X$ forming an out-tree with k leaves. Thus, there are

$$m \quad \leq \quad (k-1)(p-1) \leq (k-1)(k-2) \tag{6.1}$$

vertices of P with in-neighbors outside P and at least $q - (k-2)(k-1)$ vertices of P have both in-neighbors on P.

Let $P = u_1u_2 \dots u_q$. Suppose that there are $2(k-1)$ indices

$$i_1 < j_1 \leq i_2 < j_2 \leq \dots \leq i_{k-1} < j_{k-1}$$

such that each $u_{i_s}u_{j_s}$ is a forward arc for P. Then the arcs

$$\begin{aligned}\{u_{i_s}u_{j_s}, u_{j_s}u_{j_s+1}, \dots, u_{i_{s+1}-1}u_{i_{s+1}} :\ 1 \leq s \leq k-2\} \cup \\ \{u_{i_{k-1}}u_{j_{k-1}}\} \cup \{u_{i_s}u_{i_s+1} :\ 1 \leq s \leq k-1\}\end{aligned} \tag{6.2}$$

form an out-tree with k leaves, a contradiction.

Let f be the number of forward arcs in P. Consider the graph G whose vertices are all the forward arcs and a pair u_iu_j, u_su_r of forward arcs is adjacent in G if the intervals $[i, j-1]$ and $[s, r-1]$ of the real line intersect. Observe that G is an interval graph and, thus, a perfect graph. By the result of 6.2, the independence number of G is less than $k-1$. Thus, the chromatic number of G ($\chi(G)$) which is equal to the order g of its largest clique Q, because G is a perfect graph, is at least $f/(k-2)$. That is,

$$g = \chi(G) \geq \frac{f}{k-2},$$

since chromatic number can be viewed as the minimum number of independent sets in a partition of vertex set into independent sets.

Let $V(Q) = \{u_{i_s}u_{j_s} :\ 1 \leq s \leq g\}$ and let $h = \min\{j_s - 1 :\ 1 \leq s \leq g\}$. Observe that each interval $[i_s, j_s - 1]$ contains h. Therefore, we can form an out-tree with vertices

$$\{u_1, u_2, \dots, u_h\} \cup \{u_{j_s} :\ 1 \leq s \leq g\}$$

in which $\{u_{j_s} :\ 1 \leq s \leq g\}$ are leaves. Hence we have $\frac{f}{k-2} \leq g \leq k-1$ and, thus,

$$f \leq (k-2)(k-1). \tag{6.3}$$

Let uv be an arc of $A(D) \setminus A(P)$ such that $v \in V(P)$. There are three possibilities: (i) $u \notin V(P)$, (ii) $u \in V(P)$ and uv is forward for P, (iii) $u \in V(P)$ and uv is backward for P. By the inequalities 6.1 and 6.3 for m and f, we conclude that there are at most $2(k-2)(k-1)$ vertices on P of types (i) and (ii). These are not terminal vertices (i.e., heads) of backward arcs. Consider a path $R = v_0v_1 \dots v_r$ formed by backward arcs. Observe that the arcs $\{v_iv_{i+1} : 0 \leq i \leq r-1\} \cup \{v_jv_j^+ : 1 \leq j \leq r\}$ form an out-tree with r leaves, where v_j^+ is the out-neighbor of v_j on P. Thus, there is no path of backward arcs of length more than $k-1$.

If the in-degree of u_1 in $D[V(P)]$ is 2, then remove one of the backward arcs terminating at u_1. Observe that now the backward arcs for P form a vertex-disjoint collection of out-trees with roots at vertices that are not terminal vertices of backward arcs. Therefore, the number of the out-trees in the collection is at most $2(k-2)(k-1)$. Observe that each out-tree in the collection has at most $k-1$ leaves and thus its arcs can be decomposed into at most $k-1$ paths, each of length at most k. Hence, the original total number of backward arcs for P is at most $2k(k-2)(k-1)^2+1$. On the other hand, it is at least $(q-1)-2(k-2)(k-1)$. Thus, $(q-1)-2(k-2)(k-1) \leq 2k(k-2)(k-1)^2+1$. Combining this inequality with $q \geq n/(k-1)$, we conclude that $n \leq 2k^5$. □

Theorem 6.1 *Let D be a digraph with $\ell_s(D) = \ell(D) > 0$.*

(a) If D is an oriented graph with minimum in-degree at least 2, then $\ell_s(D) \geq (n/2)^{1/5} - 1$.

(b) If D is a digraph with minimum in-degree at least 3, then $\ell_s(D) \geq (n/2)^{1/5} - 1$.

Proof: (a) Let T be an out-branching of D. Delete some arcs arbitrarily from $A(D) \setminus A(T)$, if needed, such that the in-degree of each vertex of D becomes 2. Now the inequality $\ell_s(D) \geq (n/2)^{1/5} - 1$ follows from Lemma 6.1 and the definition of $\mathcal{L}$.

(b) Let T be an out-branching of D. Let P be the path formed in the proof of Lemma 6.1. (Note that $A(P) \subseteq A(T)$.) Delete every double arc of P, in case there are any, and delete some more arcs from $A(D) \setminus A(T)$, if needed, to ensure that the in-degree of each vertex of D becomes 2. Now the inequality $\ell_s(D) \geq (n/2)^{1/5} - 1$ follows from Lemma 6.1 and the definition of $\mathcal{L}$. □

It is not difficult to give examples showing that the restrictions on the minimum in-degrees in Theorem 6.1 are optimal. Indeed, any directed cycle C is a strong oriented graph with all in-degrees 1 for which $\ell_s(C) = 1$ and any directed double cycle D is a strong digraph with in-degrees 2 for which $\ell_s(D) = 2$ (a *directed double cycle* is a digraph obtained from an undirected cycle by replacing every edge xy with two arcs xy and yx).

6.2.2 Parameterized Algorithms for k-DMLOB and k-DMLOT

In the previous section, we gave lower bounds on $\ell(D)$ and $\ell_s(D)$ for digraphs $D \in \mathcal{L}$ with minimum in-degree at least 3. These bounds trivially imply the fixed parameter tractability of the k-DMLOB and the k-DMLOT problems for this class of digraphs. Here we extend these FPT results to digraphs in $\mathcal{L}$ for k-DMLOB and to all digraphs for k-DMLOT. We prove a decomposition theorem which either outputs an out-tree with k leaves or provides a path decomposition of the underlying undirected graph of width $O(k^2)$ in polynomial time.

Theorem 6.2 *Let D be a digraph in $\mathcal{L}$ with $\ell_s(D) > 0$. Then either $\ell_s(D) \geq k$ or the underlying undirected graph of D is of pathwidth at most $2k^2$.*

Proof: Let D be a digraph in $\mathcal{L}$ with $0 < \ell_s(D) < k$. Let us choose an out-branching T of D with p leaves. As in the proof of Lemma 6.1, we obtain a collection $\mathcal{P}$ of p $(< k)$ vertex-disjoint directed paths covering all vertices of D.

For a path $P \in \mathcal{P}$, let $W(P)$ be the set of vertices not on P which are out-neighbors of vertices on P. If $|W(P)| \geq k$, then the vertices P and $W(P)$ would form an out-tree with at least k leaves, which by the definition of $\mathcal{L}$, contradicts the assumption $\ell_s(D) < k$. Therefore, $|W(P)| < k$. We define

$$U_1 = \{v \in W(P) : P \in \mathcal{P}\}.$$

Note that

$$|U_1| \leq p(k-1) \leq (k-1)^2.$$

Let D_1 be the graph obtained from D after applying the following trimming procedure on all vertices of U_1: for every path $P \in \mathcal{P}$ and every vertex $v \in U_1 \cap V(P)$ we delete all arcs emanating out of v and directed into v except those of the path P itself. Thus for every two paths $P, Q \in \mathcal{P}$ there is no arc in D_1 that goes from P to Q.

For $P \in \mathcal{P}$ let $D_1[P]$ be the subgraph of D_1 induced by the vertices of P. Observe that P is a Hamiltonian directed path in $D_1[P]$ and the connectivity components of D_1 are the induced subgraphs of D_1 on the paths P for $P \in \mathcal{P}$.

Let $P \in \mathcal{P}$, we will show that the $pw(UN(D_1[P]))$ is bounded by $k^2 - 2k + 2$. We denote by $S[P]$ the set of vertices which are heads of forward arcs in $D_1[P]$.

We claim that $|S[P]| \leq (k-2)(k-1)$. Indeed, for each vertex $v \in S[P]$, delete all but one of the forward arcs terminating at v. Observe that the procedure has not changed the number of vertices which are heads of forward arcs. Also the number of forward arcs in the new digraph is $|S[P]|$. As in the proof of Lemma 6.1, we can show that the number of forward arcs in the new digraph is at most $(k-2)(k-1)$.

Let $D_2[P]$ be the graph obtained from $D_1[P]$ after applying the trimming procedure as before on all vertices of $S[P]$, that is, for every vertex $v \in S[P]$ we delete all arcs emanating out of v or directed into v except those of the path P.

Observe that $D_2[P]$ consists of the directed path $P = v_1v_2\dots v_q$ passing through all its vertices, together with its backward arcs. For every $j \in [q]$ let $V_j = \{v_i : i \in [j]\}$. If for some j the set V_j contained k vertices, say $\{v'_1, v'_2, \cdots, v'_k\}$, having in-neighbors in the set $\{v_{j+1}, v_{j+2}, \dots, v_q\}$, then D would contain an out-tree with k leaves formed by the path $v_{j+1}v_{j+2}\dots v_q$ together with a backward arc terminating at v'_i from a vertex on the path for each $1 \leq i \leq k$, a contradiction. Thus $vs(UN(D_2[P])) \leq k$. By Proposition 6.2, the pathwidth of $UN(D_2[P])$ is at most k. Let $(X_1, X_2, \dots, X_p)$ be a path decomposition of $UN(D_2[P])$ of width at most k. Then $(X_1 \cup S[P], X_2 \cup S[P], \dots, X_p \cup S[P])$ is a path decomposition of $UN(D_1[P])$ of width at most $k + |S[P]| \leq k^2 - 2k + 2$.

The pathwidth of a graph is equal to the maximum pathwidth of its connected components. Hence, there exists a path decomposition $(X_1, X_2, \dots, X_q)$ of $UN(D_1)$ of width at most $k^2 - 2k + 2$. Then $(X_1 \cup U_1, X_2 \cup U_1, \dots, X_q \cup U_1)$ is a path decomposition of $UN(D)$. Thus, the pathwidth of the underlying graph of D is at most $k^2 - 2k + 2 + |U_1| \leq k^2 - 2k + 2 + (k-1)^2 \leq 2k^2$. □

Theorem 6.3 *k-DMLOB is FPT for digraphs in $\mathcal{L}$ with an algorithm of time complexity $O(2^{O(k^2 \log k)} \cdot n^{O(1)})$.*

Proof: Let D be a digraph in $\mathcal{L}$ on n vertices. Using Proposition 6.1 we can test in polynomial time whether $\ell_s(D) = 0$. So from now on we assume that $\ell_s(D) > 0$. The proof of Theorem 6.2 can be easily turned into a polynomial time algorithm to either build an out-branching of D with at least k leaves or to show that $pw(UN(D)) \leq 2k^2$ and provide the corresponding path decomposition. A simple dynamic programming over the decomposition gives us an algorithm of running time $O(k^{O(k^2)} \cdot n^{O(1)})$. □

Let D be a digraph and let R_v be the set of vertices reachable from a vertex $v \in V(D)$ in D. Observe that D has an out-tree with k leaves if and only if there exists a $v \in V(D)$ such that $D[R_v]$ has an out-tree with k leaves. Notice that each $D[R_v]$ has an out-branching rooted at v. Thus, we can prove the following theorem, using the arguments in the previous proofs.

Theorem 6.4 *For a digraph D and $v \in V(D)$, let R_v be the set of vertices reachable from a vertex $v \in V(D)$ in D. Then either we have $\ell(D[R_v]) \geq k$ or the underlying undirected graph of $D[R_v]$ is of pathwidth at most $2k^2$. Moreover, one can find, in polynomial time, either an out-tree with at least k leaves in $D[R_v]$, or a path decomposition of it of width at most $2k^2$.*

To solve k-DMLOT, we apply Theorem 6.4 to all the vertices of D and then apply dynamic programming over the decomposition. This gives the following:

Theorem 6.5 *k-DMLOT is FPT for digraphs.*

6.3 Combinatorial Bounds and Algorithms - II

6.3.1 Locally Optimal Out-Trees

In this section we give faster parameterized algorithms and improved combinatorial bounds than in previous section. These improvements are based on finding locally optimal out-branchings. Given a digraph, D and an out-branching T, we call a vertex *leaf*, *link* or *branch* if its out-degree in T is 0, 1 or ≥ 2 respectively. Let $S^+_{\geq 2}(T)$ be the set of branch vertices, $S^+_1(T)$ be the set of link vertices and $L(T)$ be the set of leaves in the tree T. Let $\mathscr{P}_2(T)$ be the set of maximal paths consisting of link vertices. By $p(v)$ we denote the *parent* of a vertex v in T; $p(v)$ is the unique in-neighbor of v in T. We call a pair of vertices u and v *siblings* if they do not belong to the same path from the root r in T. We start with the following well known and easy to observe facts.

Fact 6.1 $|S^+_{\geq 2}(T)| \leq |L(T)| - 1$.

Proof: By induction on the number t of vertices of T. For $t = 1$ it is obvious. Consider a branching vertex x such that T_x (the subtree of T rooted at x) has no branching vertices but x itself. Let $T' = T - (V(T_x) - x)$. Also let $b(T) = |S^+_{\geq 2}(T)|$ and $l(T) = |L(T)|$. In T', x is a leaf. Thus, $b(T) = b(T') - 1, l(T) > l(T')$. By induction hypothesis, $b(T') < l(T')$. Thus, $b(T) < l(T)$. □

Fact 6.2 $|\mathscr{P}_2(T)| \leq 2|L(T)| - 1$.

Proof: Consider $T[S^+_1]$ which is the disjoint union of directed paths $P \in \mathscr{P}_2(T)$. With every path $P \in \mathscr{P}_2(T)$, we associate the unique out-neighbor of the last vertex of this path in T. Observe that this association is injective and the associated vertex is either a leaf or a branch vertex. Hence

$$|\mathscr{P}_2(T)| \leq |L(T)| + |S^+_{\geq 2}(T)| \leq 2|L(T)| - 1$$

from Fact 1. □

Now we define the notion of local exchange which is intensively used in our proofs.

Definition 6.1 ℓ-Arc Exchange (ℓ-AE) optimal out-branching: *An out-branching T of a directed graph D with k leaves is ℓ-AE optimal if for all arc subsets $F \subseteq A(T)$ and $X \subseteq A(D) - A(T)$ each of size ℓ, $(A(T) \setminus F) \cup X$ is either not an out-branching, or an out-branching with $\leq k$ leaves. In other words, T is ℓ-AE optimal if it can't be turned into an out-branching with more leaves by exchanging ℓ arcs.*

Let us remark, that for every fixed ℓ, an ℓ-AE optimal out-branching can be obtained in $O(nm^{\ell+2})$ time, where m and n are number of arcs and vertices respectively of the input digraph. In our proofs we use only 1-AE optimal out-branchings. We need the following simple properties of 1-AE optimal out-branchings.

Lemma 6.2 *Let T be an 1-AE optimal out-branching rooted at r in a digraph D. Then the following holds:*

(a) For every pair of siblings $u, v \in V(T) \setminus L(T)$ with $d^+_T(p(v)) = 1$, there is no arc $e = (u, v) \in A(D) \setminus A(T)$;

(b) For every pair of vertices $u, v \notin L(T)$, $d^+_T(p(v)) = 1$, which are on the same path from the root with $dist(r, u) < dist(r, v)$ there is no arc $e = (u, v) \in A(D) \setminus A(T)$ (here $dist(r, u)$ is the distance to u in T from the root r);

(c) There is no arc (v, r), $v \notin L(T)$ such that the directed cycle formed by the (r, v)-path and the arc (v, r) contains a vertex x such that $d^+_T(p(x)) = 1$.

Proof: The proof easily follows from the fact that the existence of any of these arcs contradicts the local optimality of T with respect to 1-AE. □

6.3.2 Improved Combinatorial Bounds

We start with a lemma that allows us to obtain lower bounds on $\ell_s(D)$ improving on the bound of Lemma 6.1.

Lemma 6.3 *Let D be a oriented graph of order n in which every vertex is of in-degree 2 and let D have an out-branching. If D has no out-tree with k leaves, then $n \leq 4k^3$.*

Proof: Let us assume that D has no out-tree with k leaves. Consider an out-branching T of D with $p < k$ leaves which is 1-AE optimal. Let r be the root of T.

We will bound the number n of vertices in T as follows. Every vertex of T is either a leaf, or a branch vertex, or a link vertex. By Facts 1 and 2 we already have bounds on the number of leaf and branch vertices as well as the number of maximal paths consisting of link vertices. So to get an upper bound on n in terms of k, it suffices to bound the length of each maximal path consisting of link vertices. Let us consider such a path P and let x, y be the first and last vertices of P, respectively.

The vertices of $V(T) \setminus V(P)$ can be partitioned into four classes as follows:

(a) ancestor vertices: the vertices which appear before x on the (r, x)-path of T;

(b) descendant vertices : the vertices appearing after the vertices of P on paths of T starting at r and passing through y;

(c) sink vertices: the vertices which are leaves of T but not descendant vertices;

(d) special vertices: none-of-the-above vertices.

Let $P' = P - x$, let z be the out-neighbor of y on T and let T_z be the subtree of T rooted at z. By Lemma 6.2, there are no arcs from special or ancestor vertices to the path P'. Let uv be an arc of $A(D) \setminus A(P')$ such that $v \in V(P')$. There are two possibilities for u: (i) $u \notin V(P')$, (ii) $u \in V(P')$ and uv is a backward arc with respect to P' (there are no forward arcs for P' since T is 1-AE optimal). Note that every vertex of type (i) is either a descendant vertex or a sink. Observe also that the backward arcs for P' form a vertex-disjoint collection of out-trees with roots at vertices that are not terminal vertices of backward arcs for P'. These roots are terminal vertices of arcs in which first vertices are descendant vertices or sinks.

We denote by $\{u_1, u_2, \ldots, u_s\}$ and $\{v_1, v_2, \ldots, v_t\}$ the sets of vertices on P' which have out-neighbors that are descendant vertices and sinks, respectively. Let the out-tree formed by backward arcs for P' rooted at $w \in \{u_1, \ldots, u_s, v_1, \ldots, v_t\}$ be denoted by $T(w)$ and let $l(w)$ denote the number of leaves in $T(w)$. Observe that the following is an out-tree rooted at z:

$$T_z \cup \{(in(u_1), u_1), \ldots, (in(u_s), u_s)\} \cup \bigcup_{i=1}^{s} T(u_i),$$

where $\{in(u_1), \ldots, in(u_s)\}$ are the in-neighbors of $\{u_1, \ldots, u_s\}$ respectively on T_z. This out-tree has at least $\sum_{i=1}^{s} l(u_i)$ leaves and, thus, $\sum_{i=1}^{s} l(u_i) \leq k-1$. Let us denote the subtree of T rooted at x by T_x and let $\{in(v_1), \ldots, in(v_t)\}$ be the in-neighbors of $\{v_1, \ldots, v_t\}$ on $T - V(T_x)$. Then we have the following out-tree:

$$(T - V(T_x)) \cup \{(in(v_1), v_1), \ldots, (in(v_t), v_t)\} \cup \bigcup_{i=1}^{t} T(v_i)$$

with at least $\sum_{i=1}^{t} l(v_i)$ leaves. Thus, $\sum_{i=1}^{t} l(v_i) \leq k-1$.

Consider a path $R = v_0 v_1 \ldots v_r$ formed by backward arcs. Observe that the arcs $\{v_i v_{i+1} : 0 \leq i \leq r-1\} \cup \{v_j v_j^+ : 1 \leq j \leq r\}$ form an out-tree with r leaves, where v_j^+ is the out-neighbor of v_j on P. Thus, there is no path of backward arcs of length more than $k-1$. Every out-tree $T(w)$, $w \in \{u_1, \ldots, u_s\}$ has $l(w)$ leaves and, thus, its arcs can be decomposed into $l(w)$ paths, each of length at most $k-1$. Now we can bound the number of arcs in all the trees $T(w)$, $w \in \{u_1, \ldots, u_s\}$, as follows: $\sum_{i=1}^{s} l(u_i)(k-1) \leq (k-1)^2$. We can similarly bound the number of arcs in all the trees $T(w)$, $w \in \{v_1, \ldots, v_s\}$ by $(k-1)^2$. Recall that the vertices of P' can be either terminal vertices of backward arcs for P' or vertices in $\{u_1, \ldots, u_s, v_1, \ldots, v_t\}$. Observe that $s + t \leq 2(k-1)$ since $\sum_{i=1}^{s} l(u_i) \leq k-1$ and $\sum_{i=1}^{t} l(v_i) \leq k-1$.

Thus, the number of vertices in P is bounded from above by $1 + 2(k-1) + 2(k-1)^2$. Therefore,

$$\begin{aligned} n &= |L(T)| + |S^+_{\geq 2}(T)| + |S^+_1(T)| \\ &= |L(T)| + |S^+_{\geq 2}(T)| + \sum_{P \in \mathscr{P}_2(T)} |V(P)| \\ &\leq (k-1) + (k-2) + (2k-3)(2k^2 - 2k + 1) \\ &< 4k^3. \end{aligned}$$

Thus, we conclude that $n \leq 4k^3$. □

Theorem 6.6 *Let D be a digraph in $\mathcal{L}$ with $\ell_s(D) > 0$.*

(a) If D is an oriented graph with minimum in-degree at least 2, then $\ell_s(D) \geq (n/4)^{1/3} - 1$.

(b) If D is a digraph with minimum in-degree at least 3, then $\ell_s(D) \geq (n/4)^{1/3} - 1$.

Proof: Since D is in $\mathcal{L}$, we have $\ell(D) = \ell_s(D) > 0$. Let T be an 1-AE optimal out-branching of D with maximum number of leaves.

(a) Delete some arcs from $A(D) \setminus A(T)$, if needed, such that the in-degree of each vertex of D becomes 2. Now the inequality $\ell_s(D) \geq (n/4)^{1/3} - 1$ follows from Lemma 6.3 and the fact that $\ell(D) = \ell_s(D)$.

(b) Delete some arcs arbitrarily from $A(D) \setminus A(T)$, if needed, such that the in-degree of each vertex of D becomes 2 and D is an oriented digraph (that is we delete all the double arc with respect to T). Notice that we can do this as the minimum in-degree of every vertex is at least 3. Now the inequality $\ell_s(D) \geq (n/4)^{1/3} - 1$ follows from Lemma 6.3 and the fact that $\ell(D) = \ell_s(D)$. □

Remark 6.1 *It is easy to see that Theorem 6.6 holds also for acyclic digraphs D with $\ell_s(D) > 0$.*

While we do not know whether the bounds of Theorem 6.6 are tight, we can show that no linear bounds are possible. The following result gives a lower bound for Part (b) of Theorem 6.6, but a similar result holds for Part (a) as well.

Theorem 6.7 *For each $t \geq 6$ there is a strong digraph H_t of order $n = t^2 + 1$ with minimum in-degree 3 such that $0 < \ell_s(H_t) = O(t)$.*

Proof: Let $V(H_t) = \{r\} \cup \{u_1^i, u_2^i, \dots, u_t^i \mid i \in [t]\}$ and

$$\begin{aligned} A(H_t) &= \left\{u_j^i u_{j+1}^i, u_{j+1}^i u_j^i \mid \ i \in [t], j \in \{0, 1, \dots, t-3\}\right\} \\ & \bigcup \left\{u_j^i u_{j-2}^i \mid i \in [t], j \in \{3, 4, \dots, t-2\}\right\} \\ & \bigcup \left\{u_j^i u_q^i \mid \ i \in [t], t-3 \leq j \neq q \leq t\right\}, \end{aligned}$$

where $u_0^i = r$ for every $i \in [t]$. It is easy to check that $0 < \ell_s(H_t) = O(t)$. □

6.3.3 New Decomposition Algorithms and Improved Algorithms

Theorem 6.8 *Let D be an acyclic digraph with a single vertex of in-degree zero. Then either $\ell_s(D) \geq k$ or the underlying undirected graph of D is of pathwidth at most $4k$ and we can obtain this path decomposition in polynomial time.*

Proof: Assume that $\ell_s(D) \leq k - 1$. Consider a 1-AE optimal out-branching T of D. Notice that $|L(T)| \leq k - 1$. Now remove all the leaves and branch vertices from the tree T. The remaining vertices form maximal directed paths consisting of link vertices. Delete the first vertices of all paths. As a result we obtain a collection $\mathcal{Q}$ of directed paths. Let $H = \cup_{P \in \mathcal{Q}} P$. We will show that every arc uv with $u, v \in V(H)$ is in H.

Let $P' \in \mathcal{Q}$. As in the proof of Lemma 6.3, we see that there are no forward arcs for P'. Since D is acyclic, there are no backward arcs for P'. Suppose uv is an arc of D such that $u \in R'$ and $v \in P'$, where R' and P' are distinct paths from $\mathcal{Q}$. We observe that u is either a sink or a descendent vertex for P' in T. Since R' contains no sinks of T, u is a descendent vertex, which is impossible as D is acyclic. Thus, we have proved that $pw(UN(H)) = 1$.

Consider a path decomposition of H of width 1. We can obtain a path decomposition of $UN(D)$ by adding all the vertices of $L(T) \cup S^+_{\geq 2}(T) \cup F(T)$, where $F(T)$ is the set of first vertices of maximal directed paths consisting of link vertices of T, to each of the bags of a path decomposition of H of width 1. Observe that the pathwidth of this decomposition is bounded from above by

$$|L(T)| + |S^+_{\geq 2}(T)| + |F(T)| + 1 \leq (k-1) + (k-2) + (2k-4) + 1 \leq 4k-6.$$

The bounds on the various sets in the inequality above follow from Facts 1 and 2. This proves the theorem. □

Corollary 6.1 *For acyclic digraphs, the problem k-DMLOB can solved in time $2^{O(k\log k)}n^{O(1)}$.*

Proof: The proof of Theorem 6.8 can be easily turned into a polynomial time algorithm to either build an out-branching of D with at least k leaves or to show that $pw(UN(D)) \leq 4k$ and provide the corresponding path decomposition. A simple dynamic programming over the path decomposition gives us an algorithm of running time $2^{O(k\log k)}n^{O(1)}$. □

The algorithm of Corollary 6.1 improves the $O(2^{O(k^2\log k)}n^{O(1)})$ time algorithm of Theorem 6.3 for acyclic digraphs.

The following lemma is well known, see, e.g., [62].

Lemma 6.4 *Let $T = (V, E)$ be an undirected tree and let $w : V \to \mathbb{R}^+ \cup \{0\}$ be a weight function on its vertices. There exists a vertex $v \in T$ such that the weight of every subtree T' of $T - v$ is at most $w(T)/2$, where $w(T) = \sum_{v\in V} w(v)$.*

Let D be a digraph with $\ell_s(D) = \lambda$ and let T be an out-branching of D with λ leaves. Consider the following decomposition of T (called a *β-decomposition*) which is useful in the proof of Theorem 6.9.

Assign weight 1 to all leaves of T and weight 0 to all non-leaves of T. By Lemma 6.4, T has a vertex v such that each component of $T-v$ has at most $\lambda/2+1$ leaves (if v is not the root and its in-neighbor v^- in T is a link vertex, then v^- becomes a new leaf). Let $T_1, T_2, \ldots, T_s$ be the components of $T - v$ and let $l_1, l_2, \ldots, l_s$ be the numbers of leaves in the components. Notice that $\lambda \leq \sum_{i=1}^{s} l_i \leq \lambda + 1$ (we may get a new leaf). We may assume that $l_s \leq l_{s-1} \leq \cdots \leq l_1 \leq \lambda/2+1$. Let j be the first index such that $\sum_{i=1}^{j} l_i \geq \frac{\lambda}{2} + 1$. Consider two cases: (a) $l_j \leq (\lambda+2)/4$

and (b) $l_j > (\lambda+2)/4$. In Case (a), we have

$$\frac{\lambda+2}{2} \le \sum_{i=1}^{j} l_i \le \frac{3(\lambda+2)}{4} \text{ and } \frac{\lambda-6}{4} \le \sum_{i=j+1}^{s} l_i \le \frac{\lambda}{2}.$$

In Case (b), we have $j = 2$ and

$$\frac{\lambda+2}{4} \le l_1 \le \frac{\lambda+2}{2} \text{ and } \frac{\lambda-2}{2} \le \sum_{i=2}^{s} l_i \le \frac{3\lambda+2}{4}.$$

Let $p = j$ in Case (a) and $p = 1$ in Case (b). Add to D and T a *copy* v' of v (with the same in- and out-neighbors). Then the number of leaves in each of the out-trees

$$T' = T[\{v\} \cup (\cup_{i=1}^{p} V(T_i))] \text{ and } T'' = T[\{v'\} \cup (\cup_{i=p+1}^{s} V(T_i))]$$

is between $\lambda(1+O(1))/4$ and $3\lambda(1+O(1))/4$. Observe that the vertices of T' have at most $\lambda+1$ out-neighbors in T'' and the vertices of T'' have at most $\lambda+1$ out-neighbors in T' (we add 1 to λ due to the fact that v 'belongs' to both T' and T'').

Similarly to derive T' and T'' from T, we can obtain two out-trees from T' and two out-trees from T'' in which the number of leaves are approximately between a quarter and three quarters of the number of leaves in T' and T'', respectively. Observe that after $O(\log \lambda)$ 'dividing' steps, we will end up with $O(\lambda)$ out-trees with just one leaf, i.e., directed paths. These paths contain $O(\lambda)$ copies of vertices of D (such as v' above). After deleting the copies, we obtain a collection of $O(\lambda)$ disjoint directed paths covering $V(D)$.

Theorem 6.9 *Let D be a digraph in $\mathcal{L}$ ($\ell_s(D) = 0$ or $\ell_s(D) = \ell(D)$) with $\ell_s(D) > 0$. Then either $\ell_s(D) \ge k$ or the underlying undirected graph of D is of pathwidth $O(k \log k)$.*

Proof: We may assume that $\ell_s(D) < k$. Let T be be a 1-AE optimal out-branching. Consider a β-decomposition of T. The decomposition process can be viewed as a tree $\mathcal{T}$ rooted in a node (associated with) T. The children of T in $\mathcal{T}$ are nodes (associated with) T' and T''; the leaves of $\mathcal{T}$ are the directed paths of the decomposition. The *first layer* of $\mathcal{T}$ is the node T, the *second layer* are T' and T'', the *third layer* are children of T' and T'', etc. In what follows, we do not distinguish between a node Q of $\mathcal{T}$ and the tree associated with the node. Assume that $\mathcal{T}$ has t layers. Notice that the last layer consists of (some) leaves of $\mathcal{T}$ and that $t = O(\log k)$, which was proved above ($k \leq \lambda - 1$).

Let Q be a node of $\mathcal{T}$ at layer j. We will prove that

$$pw(UN(D[V(Q)])) < 2(t - j + 2.5)k \tag{6.4}$$

Since $t = O(\log k)$, (6.4) for $j = 1$ implies that the underlying undirected graph of D is of pathwidth $O(k \log k)$.

We first prove (6.4) for $j = t$ when Q is a path from the decomposition. Let $W = (L(T) \cup S^+_{\geq 2}(T) \cup F(T)) \cap V(Q)$, where $F(T)$ is the set of first vertices of maximal paths of T consisting of link vertices. As in the proof of Theorem 6.8, it follows from Facts 1 and 2 that $|W| < 4k$. Obtain a digraph R by deleting from $D[V(Q)]$ all arcs in which at least one end-vertex is in W and which are not arcs of Q. As in the proof of Theorem 6.8, it follows from Lemma 6.2 and 1-AE optimality of T that there are no forward arcs for Q in R. Let $Q = v_1 v_2 \dots v_q$. For every $j \in [q]$, let $V_j = \{v_i : \ i \in [j]\}$. If for some j the set V_j contained k vertices, say $\{v'_1, v'_2, \cdots, v'_k\}$, having in-neighbors in the set $\{v_{j+1}, v_{j+2}, \dots, v_q\}$, then D would contain an out-tree with k leaves formed by the path $v_{j+1} v_{j+2} \dots v_q$ together with a backward arc terminating at v'_i from a vertex on the path for each $1 \leq i \leq k$, a contradiction. Thus $vs(UN(D_2[P])) \leq k$. By Proposition 6.2, the pathwidth of $UN(R)$ is at most k. Let $(X_1, X_2, \dots, X_s)$ be a path decomposition of $UN(R)$ of width at most k. Then $(X_1 \cup W, X_2 \cup W, \dots, X_s \cup W)$ is a path decomposition of $UN(D[V(Q)])$ of width less than $k + 4k$. Thus,

$$pw(UN(D[V(Q)])) < 5k \tag{6.5}$$

Now assume that we have proved (6.4) for $j = i$ and show it for $j = i-1$. Let Q be a node of layer $i-1$. If Q is a leaf of $\mathcal{T}$, we are done by (6.5). So, we may assume that Q has sons Q' and Q'' which are nodes of layer i. In the β-decomposition of T given before this theorem, we saw that the vertices of T' have at most $\lambda + 1$ out-neighbors in T'' and the vertices of T'' have at most $\lambda + 1$ out-neighbors in

T'. Similarly, we can see that (in the β-decomposition of this proof) the vertices of Q' have at most k out-neighbors in Q'' and the vertices of Q'' have at most k out-neighbors in Q' (since $k \leq \lambda - 1$). Let Y denote the set of the above-mentioned out-neighbors on Q' and Q''; $|Y| \leq 2k$. Delete from $D[V(Q') \cup V(Q'')]$ all arcs in which at least one end-vertex is in Y and which do not belong to $Q' \cup Q''$

Let G denote the obtained digraph. Observe that G is disconnected and $G[V(Q')]$ and $G[V(Q'')]$ are components of G. Thus, $pw(UN(G)) \leq b$, where

$$b = \max\{pw(UN(G[V(Q')])), pw(UN(G[V(Q'')]))\} < 2(t - i + 4.5)k \qquad (6.6)$$

Let $(Z_1, Z_2, \ldots, Z_r)$ be a path decomposition of G of width at most b. Then $(Z_1 \cup Y, Z_2 \cup Y, \ldots, Z_r \cup Y)$ is a path decomposition of $UN(D[V(Q') \cup V(Q'')])$ of width at most $b + 2k < 2(t - i + 2.5)k$. □

Similar to the proof of Corollary 6.1 and Theorems 6.3 and 6.5, we obtain the following:

Theorem 6.10 *1. For a digraph $D \in \mathcal{L}$, the problem k-DMLOB can be solved in time $2^{O(k \log^2 k)} \cdot n^{O(1)}$.*

2. k-DMLOT can be solved in time $2^{O(k \log^2 k)} \cdot n^{O(1)}$ for all digraphs.

This improves the $O(2^{O(k^2 \log k)} \cdot n^{O(1)})$ time algorithm of Theorem 6.3 for digraphs in $\mathcal{L}$.

6.4 Special Classes of Digraphs

In this section we look at some special classes of digraphs like semicomplete multipartite digraphs and semicomplete digraphs and give faster and simpler FPT algorithms for k-DMLOB and k-DMLOT problems than coming from Theorem 6.10.

6.4.1 Semicomplete Multipartite Digraphs

A digraph obtained from a complete k-partite graph by replacing every edge uv with arc uv or arc vu or both uv and vu is called a *semicomplete k-partite digraph.* A semicomplete k-partite digraph with k vertices is a *semicomplete digraph.* A tournament is a semicomplete digraph without directed 2-cycles.

Let T be an out-branching of D; $h(T)$ denotes the *height* of T, i.e., the maximum length of the path from the root of T to its leaf.

Lemma 6.5 *Let n be the number of vertices in a digraph D and let T be an out-branching of D rooted at s. Then $\ell_s(T) \geq \frac{n-1}{h(T)}$.*

Proof: Observe that, for a leaf u of T, there is a unique path P_u from s to u and let $P'_u = P_u - s$. Let $h(P_u)$ denote the length of P_u and let L denote the set of leaves of T. Observe that every leaf u can be associated with $h(P_u) - 1$ non-leaves of T excluding s. Since $\{V(P'_v) : v \in L\} = V(T) \setminus \{s\}$, we have $\ell_s(T)h(T) \geq n-1$. Thus, $\ell_s(T) \geq \frac{n-1}{h(T)}$. □

Corollary 6.2 *Let D be a semicomplete multipartite digraph of order n with at most one source. Then $\ell_s(D) \geq \frac{n-1}{4}$.*

Proof: It was proved in [141, 194] that if D has no 2-cycles, then D has an out-branching T with height at most 4. Assume that D has a 2-cycle. If D has a vertex x of out-degree $n-1$, then D has an out-branching of height 1. If D has no vertex of out-degree $n-1$, we can delete one arc from each 2-cycle of D and obtain a new semicomplete multipartite digraph D', which has no 2-cycles and which has no sources. Hence D' (and, thus, D) has an out-branching T with height at most 4. The corollary then follows from Lemma 6.5. □

We will use the following simple algorithm $\mathcal{A}$ to verify whether a digraph D has an out-branching T such that the vertices of $S \subset V(D)$ are among leaves of T:

> Delete all arcs leaving S and verify whether the remaining digraph D' has a unique source strong component; the verification can be done in time $O(|A(D')|)$ [18].

Let D be a semicomplete multipartite digraph of order n with at most one source. By Corollary 6.2, we have $\ell_s(D) \geq \frac{n-1}{4}$. Consider k-DMLOB problem. If $\frac{n-1}{4} \geq k$ then the output of k-DMLOB is Yes. Otherwise, $n \leq 4k$. Now we use the algorithm $\mathcal{A}$ for every subset of $V(D)$ of size k. There are at most $\binom{4k}{k} = O(9.4815^k)$ such subsets. We can also check whether a digraph D of order n has at most one source in $O(n^2)$ time. This implies the following:

Theorem 6.11 *k-DMLOB is $O(9.482^k + n^2)$-time solvable for semicomplete multipartite digraphs of order n, with at most one source.*

It is well known [18] that every semicomplete digraph T has an out-branching of height at most 2. Thus, by Lemma 6.5, we get $\ell_s(T) \geq \frac{n-1}{2}$. However, this bound can be significantly improved.

Theorem 6.12 *Let D be a semicomplete digraph of order n. Then $\ell_s(D) \geq n - \log_2 n$.*

Proof: Consider the following algorithm to construct an out-branching T with the leaf set L and non-leaf set I. Initially, $L = I = \emptyset$. At each iteration of the algorithm, we consider $D' = D - (L \cup I)$ and choose a vertex x with out-degree (in D') at least $(|V(D')| - 1)/2$. We add x to I and $N^+(x) \cap V(D')$ to L, where $N^+(x)$ is the set of out-neighbors of x.

To build T observe that $D[I]$ is a semicomplete digraph, and, thus, has a Hamiltonian path P [18]. By the way L was constructed, it is easy to form T from P and the vertices in L. Simple calculations show that $|L| \geq n - \log_2 n$. □

Let D be a semicomplete digraph of order n. By Theorem 6.12, $\ell_s(D) \geq n - \log_2 n$. Consider k-DMLOB problem again. If $n - \log_2 n \geq k$ then the output of k-DMLOB is Yes. Otherwise, $n \leq k + \log_2 n$. Now we again use the algorithm $\mathcal{A}$ for every subset of $V(D)$ of size k. First we note that $n \leq ck$ for any $c > 1$ in this case. And hence $n \leq k + \log_2 ck$. For $c = 2$, we can show that there are at most

$$\begin{aligned}\binom{k + \log_2 k + 1}{k} &= \binom{k + \log_2 k + 1}{\log_2 k + 1} \\ &\leq \left(\frac{e(k + \log_2 k + 1)}{\log_2 k + 1}\right)^{\log_2 k + 1} \\ &= O(k^{\log_2 k + 2.5})\end{aligned}$$

such subsets. The above arguments imply the following:

Theorem 6.13 *k-DMLOB is $O(k^{\log_2 k + 2.5} + n)$-time solvable for semicomplete digraphs of order n.*

All the improved algorithmic results in this section were based on improved bounds on $l_s(D)$. Now we give an improved lower bound on $l_s(D)$ in a directed graph consisting of two arc disjoint directed hamiltonian paths.

Theorem 6.14 *Let D be a strong oriented graph on n vertices such that D has two arc disjoint hamiltonian paths. Then $\ell_s(D) \geq (\sqrt{4+3n}-1)/3$.*

Proof: Let H_1 and H_2 be the two arc-disjoint hamiltonian paths. Let $H_1 = 12\dots n$ and let $H_2 = \pi(1)\pi(2)\dots\pi(n)$. We bipartition the arcs of H_2 as follows: $F = \{\pi(i)\pi(i+1) : \pi(i) < \pi(i+1)\}$ and $B = \{\pi(i)\pi(i+1) : \pi(i) > \pi(i+1)\}$.

Apply the following procedure. Start from the digraph $Q = H_1$. While $F \neq \emptyset$ carry out an iteration as follows: choose an arc $\pi(i)\pi(i+1) = pq \in F$, add it to Q, delete the arc $(q-1)q$ from Q, delete, from F, the arc pq as well as the arcs with $q-1$ being the head or the tail (these arcs do not necessary exist). Since at every iteration we keep the in-degree of every vertex as one except the vertex labelled 1, the resulting digraph Q has no directed cycle and hence Q is an out-branching. Observe that at each iteration we create a new leaf, which will remain as a leaf till the end of the procedure. Since in every iteration above we remove at most three arcs from F, Q has at least $|F|/3$ leaves. Therefore, $\ell_s(D) \geq |F|/3$. So if $|F| \geq (\sqrt{4+3n}-1)/3$, we are done. Otherwise, we prove the result through a different bound for $l_s(D)$.

A collection $\mathcal{G}$ of arcs in B is called a *block* if there exist $i \leq j$ such that $\mathcal{G} = \{\pi(t)\pi(t+1) : i \leq t \leq j\}$ and the arcs $\pi(i-1)\pi(i)$ and $\pi(j+1)\pi(j+2)$ are in F. Observe that there at most $|F|+1$ blocks and, thus, there is a block with at least $|B|/(|F|+1)$ arcs. Consider a block $\mathcal{G} = \{\pi(t)\pi(t+1) : i \leq t \leq j\}$. Since D has no directed 2-cycles, $\pi(t) - \pi(t+1) \geq 2$ for each $i \leq t \leq j$. Construct a new digraph Q from H_1 by adding the arcs of $\mathcal{G}$ and deleting the vertices $\{1, 2, \dots, \pi(j+1)-1\}$ and the arcs $\{(\pi(t)-1)\pi(t) : i \leq t \leq j\}$. Observe that Q is an out-tree rooted at $\pi(i)$ with at least $|\mathcal{G}| \geq |B|/(|F|+1)$ leaves. Thus, by an observation that if a strong digraph D has an out-tree with at least q leaves, then $\ell_s(D) \geq q$, we get that $\ell_s(D) \geq |B|/(|F|+1)$. Since $|F| < \sqrt{4+3n}-1)/3$, and $|B| = n-1-|F|$ the result follows. □

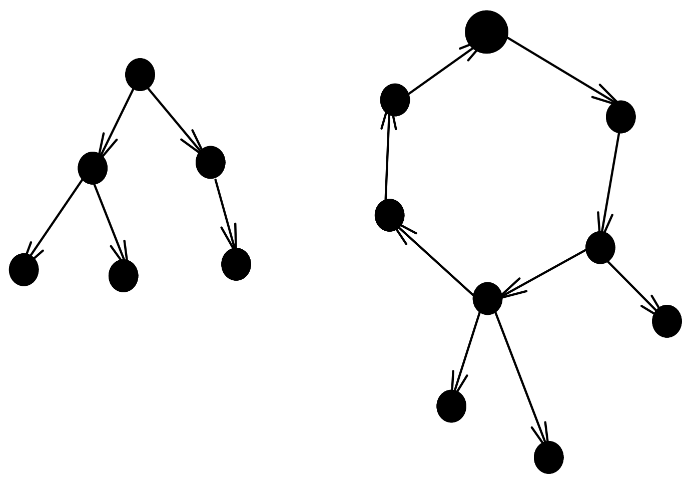

Figure 6.2: A Pseudo-Out-Branching

6.5 Pseudo-Out- Branching

In this section we give a combinatorial bound on the number of leaves or vertices of degree 0 in a weaker version of out-tree or out-branching which we call *pseudo-out-branching (POB).*

Let D be a digraph and let s be a vertex in D. A spanning subdigraph T of D is a *pseudo-out-branching (POB) rooted at* s if $d^-_T(x) = 1$ for each $x \in V \setminus \{s\}$. Notice that a component of a POB is either an out-tree rooted at s, or a directed cycle C together with a collection of out-trees $T_1, T_2, \dots, T_p$ with roots $r_1, r_2, \dots, r_p$ such that $V(C) \cap V(T_i) = \{r_i\}$ and $(V(T_i) \setminus \{r_i\}) \cap (V(T_j) \setminus \{r_j\}) = \emptyset$. See Figure 6.2. Hence if a POB does not have a component consisting of directed cycle then it is a an out-branching. Our result is as follows.

Theorem 6.15 *Let D be a digraph of order n with minimum in-degree at least 2. Then D has a POB with at least $n/3$ leaves.*

Proof: For every vertex x delete all but two arcs coming into x. Let us denote two in-neighbors of x by x', x''. Let the resulting digraph be H. Construct an undirected graph G as follows: $V(G) = V(H)$ and $E(G) = \{xy : \ xz, yz \in A(H) \text{ for some } z\}$. Find a maximum independent set I in G. Now for every vertex x of H, delete the arc $x'x$ from H if $x' \in I$, and delete $x''x$ from H, otherwise. We have obtained a new digraph T' in which the in-degree of each vertex equals 1. By deleting one outgoing arc from T' we get a POB T with at least $|I|$ leaves.

Clearly, T has at least as many leaves as T'. Let us estimate the number of leaves in T'. By the definition of I and the way we constructed T', every vertex $x \in I$ is a leaf in T'. It is known (see pages 91-92 of [10]) that $|I| \geq \sum_{v \in V(G)} \frac{1}{avd(G)+1}$, where $avd(G)$ is the average degree of G. Thus, T has at least $n/3$ leaves. □

Remark 6.1 *The bound of Theorem 6.15 is (almost) tight. Consider a set of vertex-disjoint triangles. Replace every edge xy by the arcs xy, yx. Clearly, the maximum number of leaves in a POB of the obtained digraph is* $1 + n/3$.

6.6 Conclusion

We have shown that every digraph $D \in \mathcal{L}$ with $\ell_s(D) > 0$ of order n and with minimum in-degree at least 3 contains an out-branching with at least $(n/4)^{1/3} - 1$ leaves. Combining the ideas in the proof of this combinatorial result with the fact that the problem of deciding whether a given digraph in $\mathcal{L}$ has an out-branching with at least k leaves can be solved efficiently for digraphs of pathwidth at most $O(k \log k)$ we have shown that the k-DMLOB problem for digraphs in $\mathcal{L}$ as well as the k-DMLOT problem for general digraphs are fixed parameter tractable. The parameterized complexity of the k-DMLOB problem for all digraphs remains open.

7 Complexity of Some Colorful Problems Parameterized by Treewidth

We tend to think that "all" (or almost all) combinatorial problems are easy for graphs of bounded treewidth. For an example in the last chapter, we showed that the k-DMLOB and the k-DMLOT problems are fixed parameter tractable on graphs of bounded treewidth. But in the case of structured coloring problems, it is not true. Here, we show a few variations of coloring problem to be W[1]-hard when parameterized by treewidth of the input graph.

In this chapter, we study the computational complexity of List Coloring and Equitable Coloring problems for graphs of bounded treewidth, in the framework of parameterized complexity. We show that these problems are intractable when we take the parameter to be the treewidth bound t. These results are in contrast to the other results of this part of the thesis where we have shown the problems to be fixed parameter tractable.

Our reductions for List Coloring and Equitable Coloring problems are based on recently developed methodology for proving a problem to be W-hard. This methodology is known as Multicolor Clique, vertex and edge representation strategy. While the List Coloring problem is based on the Multicolor Clique vertex representation strategy, the Equitable Coloring problem is based on the Multi Color Clique edge representation strategy,

7.1 List Coloring : Multicolor Clique Vertex Representation

The problem of LIST COLORING is defined as follows:

> LIST COLORING: A graph $G = (V, E)$ and for each vertex $v \in V$, a list $L(v)$ of permitted colors. Is there a proper vertex coloring c with $c(v) \in L(v)$ for each v ?

There is a simple reduction to the LIST COLORING (when parameterized by the treewidth t) from the MULTICOLOR CLIQUE problem which is defined as follows.

> MULTICOLOR CLIQUE : The problem takes as input a graph G together with a proper k-coloring of the vertices of G. The question is whether there is a k-clique in G consisting of exactly one vertex of each color.

The MULTICOLOR CLIQUE problem is known to be $W[1]$-complete [104] (by a simple reduction from the ordinary CLIQUE). Starting a reduction from colored version of different problems has many advantages and gives us a schematic way to design gadgets. Let $V[i]$ be the set of vertices in the color class i and $E[i, j]$ be the set of edges between color class i and j. Then we can assume that $|V[i]| = N$ for all i, and that $|E[i, j]| = M$ for all $i < j$, that is, we can assume that the vertex color classes of G, and also the edge sets between them, have uniform sizes. For a simple justification of this assumption, we can reduce MULTICOLOR CLIQUE to itself, taking a union of $k!$ disjoint copies of G, one for each permutation of the color set.

Now we show that the LIST COLORING problem on graphs of treewidth t is W[1]-hard when parameterized by treewidth.

Given the source instance G of MULTICOLOR CLIQUE problem, we construct an instance G' of LIST COLORING that admits a proper choice of color from each list if and only if the source instance G has a multicolor k-clique. The colors on the lists of vertices in G' are in one to one correspondence with the vertices of G. There are k vertices $v[i]$ in G', $i = 1, ..., k$, one for each color class of G, and the list assigned to $v[i]$ consists of the colors corresponding to the vertices in G of color i that is $L_{v[i]} = \{V[i]\}$. For $i \neq j$, there are various vertices of degree two in G'

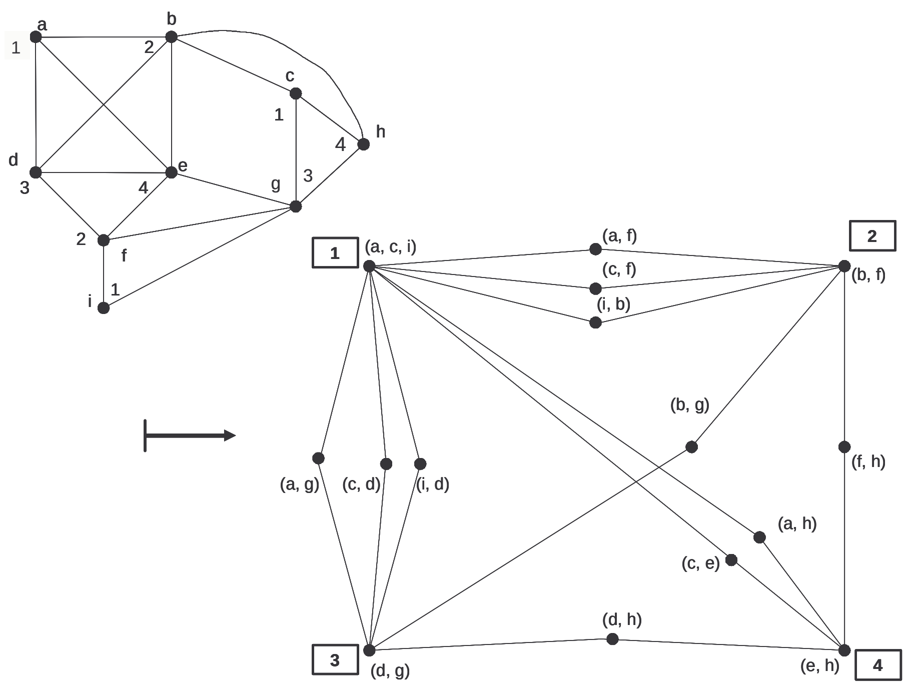

Figure 7.1: Example of the reduction from Multicolor Clique to List Coloring

adjacent to $v[i]$ and $v[j]$, each having a list of size 2. There is one such degree two vertex in G' adjacent to $v[i]$ and $v[j]$ for each pair x, y of *nonadjacent* vertices in G, where x has color i and y has color j. This vertex is labeled $v_{i,j}[x, y]$ and has $\{x, y\}$ as its list. This completes the construction. As example of the reduction is shown in Figure 7.1. The figure shows, for the parameter value $k = 4$.

The treewidth of G' is bounded by k as the graph obtained after removing the vertices $v[i]$, $1 \leq i \leq k$, from G' is an empty graph and hence has treewidth 0 (anyway it is well known that degree 2 vertices do not increase the treewidth of a graph). Now, if G has a multicolor clique K then we can easily list color G'. Assign $v[i]$ with the vertex (color in G') corresponding to the color class $V[i]$ in the multicolor clique K. Now it is easy to see that every degree 2 vertex in G' has at least one color free in its list. For the other direction, we show that the vertices of G, corresponding to the colors assigned to $v[i]$'s in a list coloring of G', forms a clique. This follows since two vertices u and v of G belonging to different color classes do not appear together on a list of some degree 2 vertices in G' if and only if they have an edge (u, v) between them in G. This results in the following theorem.

Theorem 7.1 List Coloring *problem parameterized by treewidth is* $W[1]$*-hard.*

7.2 Equitable Coloring : Multi Color Clique Edge Representation

The problem of equitable coloring is defined as follows.

> Equitable Coloring Problem (ECP): Given an input graph G and a positive r, does there exist a proper coloring of G with r colors such that for any two color class, V_i and V_j, $||V_i| - |V_j|| \leq 1$.

The notion of equitable coloring seems to have been first introduced by Meyer in 1973, where an application to scheduling garbage trucks is described [182]. Recently, Bodlaender and Fomin have shown that determining whether a graph of treewidth at most t admits an equitable coloring, can be solved in time $O(n^{O(t)})$ [34].

We consider the parameterized complexity of Equitable Coloring Problem in graphs with treewidth bounded by t. We show that ECP parameterized by (t, r), where t is the treewidth bound, and r is the number of color classes, is W[1]-hard.

In this section we show a reduction based on a methodology which is sometimes termed an *edge representation strategy* for the parameterized reduction from Multicolor Clique to Equitable Coloring problem. This strategy is very basic and is useful for many reduction. Consider that the instance $G = (V, E)$ of Multicolor Clique has its vertices colored by the integers $1, ..., k$. Let $V[i]$ denote the set of vertices of color i, and let $E[i, j]$, for $1 \leq i < j \leq k$, denote the set of edges $e = uv$, where $u \in V[i]$ and $v \in V[j]$. We also assume that $|V[i]| = N$ for all i, and that $|E[i, j]| = M$ for all $i < j$, that is, the vertex color classes of G, and also the edge sets between them, have uniform sizes. In this methodology our basic encoding gadgets correspond to edges which we call *edge gadget.* We generally have three kind of gadgets which are engineered together to get an overall reduction gadget for the problem.

Selection Gadget: This gadget job is to select exactly one edge gadget among edge gadgets corresponding to edges between any two color classes $V[i]$ and $V[j]$.

Coherence Gadget: This gadget makes sure that the edge gadgets selected among edge gadgets corresponding to edges emanating out from a particular color class $V[i]$ has a vertex in common in $V[i]$. That is all the edges corresponding to selected edge gadgets emanates from the same vertex in $V[i]$.

Match Gadget: This gadget ensures that if we have selected an edge gadget corresponding to an edge (u, v) between $V[i]$ and $V[j]$ then the edge gadget selected between $V[j]$ and $V[i]$ corresponds to (v, u).

In what follows next we show how to adhere to this strategy and form gadgets in the context of reduction from MULTICOLOR CLIQUE to EQUITABLE COLORING PROBLEM in graphs with bounded treewidth.

To show the desired reduction, we introduce two more general problems. List analogues of equitable coloring have been previously studied by Kostochka, et al. [169].

LIST EQUITABLE COLORING PROBLEM (LECP): Given an input graph $G = (V, E)$, lists L_v of colors for every vertex $v \in V$ and a positive integer r; does there exist a proper coloring f of G with r colors that for every vertex $v \in V$ uses a color from its list L_v such that for any two color classes, V_i and V_j of the coloring f, $||V_i| - |V_j|| \leq 1$?

NUMBER LIST COLORING PROBLEM (NLCP): Given an input graph $G = (V, E)$, lists L_v of colors for every vertex $v \in V$, a function $h : \cup_{v \in V} L_v \rightarrow \mathbb{N}$, associating a number to each color, and a positive integer r; does there exist a proper list coloring f of G with r colors such that $|\{v \in V \mid f(v) = c\}| \leq h(c)$?

Our main effort is in the reduction of the MULTICOLOR CLIQUE problem to NLCP.

We will use the following sets of colors in our construction of an instance of NLCP:

1. $\mathcal{S} = \{\sigma[i,j] : 1 \leq i \neq j \leq k\}$
2. $\mathcal{S}' = \{\sigma'[i,j] : 1 \leq i \neq j \leq k\}$
3. $\mathcal{T} = \{\tau_i[r,s] : 1 \leq i \leq k,\ 1 \leq r < s \leq k, r \neq i, s \neq i\}$
4. $\mathcal{T}' = \{\tau_i'[r,s] : 1 \leq i \leq k,\ 1 \leq r < s \leq k, r \neq i, s \neq i\}$
5. $\mathcal{E} = \{\epsilon[i,j] : 1 \leq i < j \leq k\}$
6. $\mathcal{E}' = \{\epsilon'[i,j] : 1 \leq i < j \leq k\}$

Note that $|\mathcal{S}| = |\mathcal{S}'| = 2\binom{k}{2}$, that is, there are distinct colors $\sigma[2,3]$ and $\sigma[3,2]$, etc. In contrast, the colors $\tau_i[r,s]$ are only defined for $r < s$.

We associate with each vertex and edge of G a pair of (unique) *identification numbers*. The *up-identification number* $v[up]$ for a vertex v should be in the range $[n^2+1, n^2+n]$, if G has n vertices and it could be chosen arbitrarily, but uniquely. Similarly, the *up-identification number* $e[up]$ of an edge e of G can be assigned (arbitrarily, but uniquely) in the range $[2n^2+1, 2n^2+m]$, assuming G has m edges.

Choose a suitably large positive integer Z_0, for example $Z_0 = n^3$, and define the *down-identification number* $v[down]$ for a vertex v to be $Z_0 - v[up]$, and similarly for the edges e of G, define the *down-identification number* $e[down]$ to be $Z_0 - e[up]$.

Choose a second large positive integer, $Z_1 >> Z_0$, for example, we may take $Z_1 = n^6$.

Next we describe various gadgets and the way they are combined in the reduction. First we describe the gadget which encodes the *selection* of the edge going between two particular color classes in G. In other words, we will think of the representation of a k-clique in G as involving the selection of edges (with each edge selected twice, once in each direction) between the color classes of vertices in G, with gadgets for *selection*, and to check two things: (1) that the selections in opposite color directions match, and (2) that the edges chosen from color class $V[i]$ going to $V[j]$ (for various $j \neq i$) all emanate from the same vertex in $V[i]$.

There are $2\binom{k}{2}$ groups of gadgets, one for each pair of color indices $i \neq j$. If $1 \leq i < j \leq k$, then we will refer to the gadgets in the group $\mathcal{G}[i,j]$ as *forward gadgets*, and we will refer to the gadgets in the group $\mathcal{G}[j,i]$ as *backward gadgets*.

If $e \in E[i,j]$, then there is one forward gadget corresponding to e in the group $\mathcal{G}[i,j]$, and one backward gadget corresponding to e in the group $\mathcal{G}[j,i]$. The construction of these gadgets is described as follows.

The forward gadget corresponding to $e = uv \in E[i,j]$.
The gadget has a root vertex $r[i,j,e]$, and consists of a tree of height 2. The list assigned to this root vertex contains two colors: $\sigma[i,j]$ and $\sigma'[i,j]$. The root vertex has Z_1+1 children, and each of these is also assigned the two-element list containing the colors $\sigma[i,j]$ and $\sigma'[i,j]$. One of the children vertices is distinguished, and has $2(k-1)$ groups of further children:

- $e[up]$ children assigned the list $\{\sigma'[i,j], \epsilon[i,j]\}$.
- $e[down]$ children assigned the list $\{\sigma'[i,j], \epsilon'[i,j]\}$.
- For each r in the range $j < r \leq k$, $u[up]$ children assigned the list $\{\sigma'[i,j], \tau_i[j,r]\}$.
- For each r in the range $j < r \leq k$, $u[down]$ children assigned $\{\sigma'[i,j], \tau_i'[j,r]\}$.
- For each r in the range $1 \leq r < j$, $u[down]$ children assigned $\{\sigma'[i,j], \tau_i[r,j]\}$.
- For each r in the range $1 \leq r < j$, $u[up]$ children assigned the list $\{\sigma'[i,j], \tau_i'[r,j]\}$.

The backward gadget corresponding to $e = uv \in E[i,j]$.
The gadget has a root vertex $r[j,i,e]$, and consists of a tree of height 2. The list assigned to this root vertex contains two colors: $\sigma[j,i]$ and $\sigma'[j,i]$. The root vertex has Z_1+1 children, and each of these is also assigned the two-element list containing the colors $\sigma[j,i]$ and $\sigma'[j,i]$. One of the children vertices is distinguished, and has $2k$ groups of further children:

- $e[up]$ children assigned the list $\{\sigma'[j,i], \epsilon'[i,j]\}$.
- $e[down]$ children assigned the list $\{\sigma'[j,i], \epsilon[i,j]\}$.
- For each r in the range $i < r \leq k$, $v[up]$ children assigned the list $\{\sigma'[j,i], \tau_j[i,r]\}$.
- For each r in the range $i < r \leq k$, $v[down]$ children assigned $\{\sigma'[j,i], \tau_j'[i,r]\}$.
- For each r in the range $1 \leq r < i$, $v[down]$ children assigned $\{\sigma'[j,i], \tau_j[r,i]\}$.
- For each r in the range $1 \leq r < i$, $v[up]$ children assigned the list $\{\sigma'[j,i], \tau_j'[r,i]\}$.

The numerical targets (h function) .

1. For all $c \in (\mathcal{T} \cup \mathcal{T}')$, $h(c) = Z_0$.
2. For all $c \in (\mathcal{E} \cup \mathcal{E}')$, $h(c) = Z_0$.
3. For all $c \in \mathcal{S}$, $h(c) = (M-1)(Z_1+1)+1$.
4. For all $c \in \mathcal{S}'$, $h(c) = (M-1)+(Z_1+1)+(k-1)(M-1)Z_0$.

That completes the formal description of the reduction from Multicolor Clique to NLCP. We turn now to some motivating remarks about the design of the reduction.

Remarks on the colors, their numerical targets, and their role in the reduction.

(**1**). There are $2\binom{k}{2}$ groups of gadgets. Each edge of G gives rise to two gadgets. Between any two color classes of G there are precisely M edges, and therefore $M \cdot \binom{k}{2}$ edges in G in total. Each group of gadgets therefore contains M gadgets. The gadgets in each group have two "helper" colors. For example, the group of gadgets $\mathcal{G}[4,2]$ has the helper colors $\sigma[4,2]$ and $\sigma'[4,2]$. The role of the gadgets in this group is to indicate a choice of an edge going *from* a vertex in the color class $V[4]$ of G *to* a vertex in the color class $V[2]$ of G. The role of the $2\binom{k}{2}$ groups of gadgets is to represent the selection of $\binom{k}{2}$ edges of G that form a k-clique, with each edge chosen twice, once in each direction. If $i < j$ then the choice is represented by the coloring of the gadgets in the group $\mathcal{G}[i,j]$, and these are the *forward* gadgets of the edge choice. If $j < i$, then the gadgets in $\mathcal{G}[i,j]$ are *backward* gadgets (representing the edge selection in the opposite direction, relative to the ordering of the color classes of G). The numerical targets for the colors in $\mathcal{S} \cup \mathcal{S}'$ are chosen to force exactly one edge to be selected (forward or backward) by each group of gadgets, and to force the gadgets that are colored in a way that indicates the edge was not selected into being colored in a particular way (else the numerical targets cannot be attained). The numerical targets for these colors are complicated, because of this role (which is asymmetric between the pair of colors $\sigma[i,j]$ and $\sigma'[i,j]$).

(**2**). The colors in $\mathcal{T} \cup \mathcal{T}'$ and $\mathcal{E} \cup \mathcal{E}'$ are organized in symmetric pairs, and each pair is used to transmit (and check) information. Due to the enforcements alluded to above, each "selection" coloring of a gadget (there will be only one possible in each group of gadgets) will force some number of vertices to be colored with these pairs of colors, which can be thought of as an information transmission. For example, when a gadget in $\mathcal{G}[4,2]$ is colored with a "selection" coloring, this indicates that the edge from which the gadget arises is selected as the edge *from* the color class $V[4]$ of G, *to* the color class $V[2]$. There is a pair of colors that handles the information transmission concerning *which edge is selected* between the groups $\mathcal{G}[2,4]$ and $\mathcal{G}[4,2]$. (Of course, something has to check that the edge selected in one direction, is the same as the edge selected in the other direction.) There is something elegant about the dual-color transmission channel for this information. Each vertex and edge has two unique identification numbers, "up" and "down", that sum to Z_0. To continue the concrete example, $\mathcal{G}[4,2]$ uses the (number of vertices colored by the) pair of colors $\epsilon[2,4]$ and $\epsilon'[2,4]$ to communicate to $\mathcal{G}[2,4]$ about the edge selected. The signal from one side consists of $e[up]$ vertices colored $\epsilon[2,4]$ and $e[down]$ vertices colored $\epsilon'[2,4]$. The signal from the other side consists of $e[down]$ vertices colored $\epsilon[2,4]$ and $e[up]$ vertices colored $\epsilon'[2,4]$. Thus the numerical targets for these colors allow us to check whether the same edge has been selected in each direction (if each color target of Z_0 is met). There is the additional advantage that the *amount* of signal in each direction is the same: in each direction a total of Z_0 colored vertices, with the two paired colors, constitutes the signal. This means that, modulo the discussion in (1) above, when an edge is *not* selected, the corresponding non-selection coloring involves uniformly the same number (i.e., Z_0) of vertices colored "otherwise" for each of the $(M-1)$ gadgets colored in the non-selection way: this explains (part of) the $(k-1)(M-1)Z_0$ term in (4) of the numerical targets.

(**3**). In a similar manner to the communication task discussed above, each of the $k-1$ groups of gadgets $\mathcal{G}[i, \_]$ need to check that each has selected an edge *from* $V[i]$ that originates at the same vertex in $V[i]$. Hence there are pairs of colors that provide a communication channel similar to that in (2) for this information. This role is played by the colors in $\mathcal{T} \cup \mathcal{T}'$. (Because of the bookkeeping issues, this becomes somewhat intricate in the formal definition of the reduction.)

The above remarks are intended to aid an intuitive understanding of the reduction. We now return to a more formal argument.

Claim 7.1 *If G has a k-multicolor clique, then G' is a yes-instance to NLCP.*

Proof: The proof of this claim is relatively straightforward. The gadgets corresponding to the edges of a k-clique in G are colored in a manner that indicates "selected" (for both the forward and the backward gadgets) and all other gadgets are colored in manner that indicates "non-selected". The coloring that corresponds to "selected" colors the root vertex with the color $\sigma[i, j]$, and this forces the rest of the coloring of the gadget. The coloring that corresponds to "non-selected" colors the root vertex with the color $\sigma'[i, j]$. In this case the coloring of the rest of the gadget is not entirely forced, but if the grandchildren vertices of the gadget are also colored with $\sigma'[i, j]$, then all the numerical targets will be met. □

Claim 7.2 *Suppose that* Γ *is a list coloring of* G' *that meets all the numerical targets. Then in each group of gadgets, exactly one gadget is colored in a way that indicates "selection".*

Proof: We argue this as follows. There cannot be two gadgets in any group colored in the "selection" manner, since this would make it impossible to meet the numerical target for a color in $\mathcal{S}$. If no gadget is colored in the "selection" manner, then again the targets cannot be met for the colors in $\mathcal{S} \cup \mathcal{S}'$ used in the lists for this group of gadgets. □

Claim 7.3 *Suppose that* Γ *is a list coloring of* G' *that meets all the numerical targets. Then in each group of gadgets, every gadget that is not colored in a way that indicates "selection" must have all of its grandchildren vertices colored with the appropriate color in* $\mathcal{S}'$.

Proof: This claim follows from Claim 7.2, noting that the numerical targets for the $\mathcal{S}'$ colors cannot be met unless this is so. □

It follows from Claims 7.2 and 7.3, that if Γ is a list coloring of G' that meets all the numerical targets, then in each group of gadgets, exactly one gadget is colored in the "selection" manner, and all other gadgets are colored in a completely determined "nonselection" manner. Each "selection" coloring of a gadget produces a numerical signal (based on vertex and edge identification numbers) carried by the colors in $\mathcal{T} \cup \mathcal{T}'$ and $\mathcal{E} \cup \mathcal{E}'$, with two signals per color. The target of Z_0 for these colors can only be achieved if the selection colorings indicate a clique in G.

Theorem 7.2 *NLCP is W[1]-hard for trees, parameterized by the number of colors that appear on the lists.*

The reduction from NLCP to LECP is almost trivial, achieved by padding with isolated vertices having single-color lists.

The reduction from LECP to ECP is described as follows. Create a clique of size r, the number of colors occurring on the lists, and connect the vertices of this clique to the vertices of G' in a manner that enforces the lists. Since G' is a tree, the treewidth of the resulting graph is at most r. We have:

Theorem 7.3 Equitable Coloring *is $W[1]$-hard, parameterized by treewidth.*

Part III

Exact Exponential Time Algorithms

8 Efficient Exact Algorithms through FPT Algorithms

In the first part of the thesis we gave fixed parameter tractable algorithms for various problems. In this chapter we explore the possibility of obtaining a non trivial exact algorithm for the optimization version of various problems using parameterized algorithm for the same problem.

We develop a simple technique in Section 8.1 by which we can use a parameterized algorithm of time complexity $O^*((4-\epsilon)^k)$, where k is the parameter and $\epsilon > 0$, to obtain an exact algorithm of time complexity $O^*((2-\eta)^n)$, $\eta > 0$. This technique is based on a careful use of the parameterized algorithm for certain values of the parameter and brute-force for other values. In Section 8.2, we give several applications of this technique which include algorithms for Odd Cycle Transversal, Feedback Vertex Set in tournaments to name a few. Finally we conclude in Section 8.3 with some remarks.

8.1 Using FPT algorithms to design exact algorithms

In this section, we give a general technique of designing exact algorithms using parameterized algorithms as a subroutine and apply it to several problems including the Odd Cycle Transversal and Feedback Vertex Set. Let Q be an NP-optimization problem and suppose that its parameterized version (Q, k) is fixed parameter tractable. An instance of the parameterized version is a tuple $\langle I, k\rangle$, where I is an instance of Q and k is an integer. The question is to decide whether I has a solution of size $\geq k$, if Q is a maximization problem, and of size $\leq k$, if Q is a minimization problem. Let us also suppose that the FPT algorithm $\mathscr{A}$ for (Q, k) has a time complexity of the form $O^*(c^k)$, where k is the parameter, and c is a constant. This algorithm $\mathscr{A}$ immediately gives us an exact algorithm for Q with time complexity $O^*(c^n)$, where n is an upper bound on the optimum solution size. What is interesting is that the FPT algorithm can actually give us an exact algorithm for Q with time complexity $O^*(d^n)$, where $d < c$. Moreover, if $c < 4$ then we will show that $d < 2$.

This fact has an interesting consequence. There are many optimization problems such as Max Independent Set, Min Vertex Cover, Min Feedback Vertex Set which have trivial brute-force enumeration algorithms of time complexity 2^n, where n is the size of the vertex set. If the parameterized versions of any of these problems is solvable in time $O^*(c^k)$, where $c < 4$ then we immediately obtain exact algorithms for the optimization version of these problems which are better than the trivial brute-force algorithms. We will show that this technique simplifies exact algorithms for many optimization problems and for some gives the best known exact algorithm.

Our algorithm makes clever use of the FPT algorithm $\mathscr{A}$ and brute-force enumeration. Consider a problem such as Odd Cycle Transversal. Had we used brute-force throughout, then the time complexity would have been $O^*(\sum_{i=0}^{n}\binom{n}{i}) = O^*(2^n)$. It is well known that the function $\binom{n}{i}$ increases with increasing i, attains a maximum at $i = n/2$, and then decreases. Also, it is symmetric in that $\binom{n}{i} = \binom{n}{n-i}$. Brute-force pushes the time complexity to $O^*(2^n)$ because it is costlier to search exhaustively when i is near $n/2$, since $\binom{n}{n/2} \approx 2^n$. Therefore, if we adopt the strategy of using brute-force only for those values of i which are far removed from $n/2$ and using the FPT algorithm $\mathscr{A}$ for the remaining i values (that is, those near $n/2$), then we might end up with an exponential-time complexity better than that of $\mathscr{A}$. And indeed we do. Our algorithm is given in Figure 8.1. For simplicity the algorithm considers minimization problems only. For maximization problems we can modify the algorithm to output the largest i for which there exists a solution.

Suppose the FPT algorithm $\mathscr{A}$ for Q takes $O^*(c^k)$ time, where c is some con-

```
Algorithm Exact($Q$,$\mathscr{A}$,$c$)
($Q$ is a minimization problem and $\mathscr{A}$ is the FPT algorithm that
solves its parameterized version in time $O^*(c^k)$, where $c$ is a constant
and $k$ is a parameter. Here $n$ is an upper bound on the optimum
solution size as well as on the size of the universe $U$. Any solution
is a subset of $U$.)
Compute the largest $\lambda$ such that $c^{\lfloor n\lambda \rfloor} \leq \sum_{i=\lfloor n\lambda \rfloor+1}^{n} \binom{n}{i}$.

 for $i = 1$ to $\lfloor \lambda n \rfloor$
     use the FPT algorithm $\mathscr{A}$ for $Q$ to check whether there is a solu-
     tion $S$ of size $i$; if yes output $S$ and halt.
 for $i = \lfloor \lambda n \rfloor + 1$ to $n$
     for every subset $S$ of size $i$ of the universe $U$, try whether $S$ is a
     solution; if yes, then output $S$ and halt.
```

Figure 8.1: Algorithm Exact()

stant. Then from the description of Algorithm Exact, it is easy to observe that its time complexity is upper bounded by following:

$$O^*\left(c^{\lfloor \lambda n \rfloor}\right) = O^*\left(\sum_{i=\lfloor n\lambda \rfloor+1}^{n} \binom{n}{i}\right) \tag{8.1}$$

Now suppose that the trivial brute-force algorithm for Q has time complexity $O^*(2^n)$. We show that if we want Algorithm Exact to beat this trivial time bound then we must have $c < 4$. We need a lemma.

Lemma 8.1 *Let $\frac{1}{2} < \lambda < 1$. Then $\binom{n}{n-\lambda n}$ is bounded by d^n, where d is some constant < 2.*

Proof: We know that

$$\binom{n}{n-\lambda n} = \binom{n}{\lambda n} \leq \frac{n^n}{(\lambda n)^{\lambda n}((1-\lambda)n)^{(1-\lambda)n}} = \left(\left(\frac{1}{\lambda}\right)^{\lambda}\left(\frac{1}{1-\lambda}\right)^{1-\lambda}\right)^n$$

One can easily verify using calculus that the function

$$h(\lambda) = \left(\frac{1}{\lambda}\right)^{\lambda} \left(\frac{1}{1-\lambda}\right)^{1-\lambda} \quad (0 < \lambda < 1)$$

attains a maximum of 2 at $\lambda = 1/2$. At other points in the interval $(\frac{1}{2}, 1)$ it has a value less than 2. This proves the claim. □

Now it is easy to see that

$$O^*\left(\sum_{i=\lfloor n\lambda \rfloor+1}^{n} \binom{n}{i}\right) = \begin{cases} \Omega\left(2^n\right) & \text{if } \lambda \leq 1/2 \\ \\ O^*\left(\binom{n}{n-\lambda n}\right) & \text{if } \lambda > 1/2 \end{cases}$$

If $c = 4 - \epsilon$ in algorithm of Figure 8.1, for some $0 < \epsilon < 4$, then we must have $\lambda > 1/2$. To see this assume that $\lambda \leq 1/2$. We would then have:

$$(4-\epsilon)^{\lfloor \lambda n \rfloor} \approx \sum_{i=\lfloor n\lambda \rfloor+1}^{n} \binom{n}{i} = \Omega(2^n).$$

This is clearly impossible since $(4-\epsilon)^{\lambda n} = o(2^n)$ when $\lambda \leq 1/2$.

Now using Lemma 8.1 and the above discussions, we obtain the following theorem.

Theorem 8.1 *Let Q be an* NP*-optimization problem such that for every instance I of Q there is a polynomial time computable universe U of size, say n, such that an optimum solution of I is a subset of U. Suppose that the parameterized version of Q is* FPT *with an algorithm of time complexity $O^*(c^k)$, then there is an exact algorithm for Q with time complexity $O^*(d^n)$, where $d = c^\lambda$ and λ (< 1) is the largest value such that $c^{\lfloor n\lambda \rfloor} \leq \sum_{i=\lfloor n\lambda \rfloor+1}^{n} \binom{n}{i}$. In particular, $c < 4$ implies $d = c^\lambda < 2$.*

8.2 Applications

In this section, we apply the algorithm developed in Section 8.1 to various problems and obtain exact algorithms with nontrivial worst-case time bounds. The problems for which we give efficient exact algorithms include the ODD CYCLE

Transversal problem in general undirected graphs, the 3 and 4-Hitting Set problems, and the Feedback Arc (Vertex) Set problem in tournaments. Some of these results are new and some of them are given here to show the applicability of Theorem 8.1 and Algorithm Exact().

8.2.1 Odd Cycle Transversal in General Graphs

The Odd Cycle Transversal problem, finding the minimum number of vertices whose deletion makes the graph bipartite, can be solved exactly in $O^*(2^{|V|})$ time. Reed, Kaleigh, and Vetta [202] have recently given an FPT algorithm for the parameterized version of this problem with running time $O(3^k kmn)$. If we use their FPT algorithm directly to solve the optimization version of the problem we will take time $O^*(3^n)$ which is worse than that taken by the trivial exponential-time algorithm. However, if we use the algorithm in Figure 8.1 with $c = 3$, we obtain $\lambda = 0.6091$ and get a running time of $O^*(1.9526^n)$. We therefore have the following theorem.

Theorem 8.2 *The* Odd Cycle Transversal *problem can be solved in time $O^*(1.9526^n)$ on a graph on n vertices.*

An algorithm for this problem with time complexity $O^*(1.4908^n)$ will be presented in the next chapter.

8.2.2 Odd Cycle Transversal in 3-Colorable and Max Degree 3 Graphs

In this section, we relate the notions of odd cycle transversal and the chromatic number of a graph. We use this to show that $n/3$ is an upper bound on the size of a minimum odd cycle transversal in 3-colorable and maximum degree 3 graphs.

Lemma 8.2 *Let G be a graph on n vertices with chromatic number $k \geq 2$. Then G has an odd cycle transversal of size at most $\frac{(k-2)n}{k}$.*

Proof: Consider a proper k-coloring of G. Let $C_1, C_2, \ldots, C_k$ be the k color classes ordered so that $|C_i| \leq |C_{i+1}|$, $1 \leq i \leq k$. Clearly, if we remove the vertices in the first $k-2$ color classes from G, we will be left with a bipartite graph and hence

$C_1 \cup C_2 \cdots \cup C_{k-2}$ is an odd cycle transversal. We claim that $\sum_{i=1}^{k-2} |C_i| \leq \frac{n(k-2)}{k}$. Suppose not. Then $\sum_{i=1}^{k-2} |C_i| > \frac{n(k-2)}{k}$. Since

$$\frac{1}{k-2}\sum_{i=1}^{k-2} |C_i| \leq |C_{k-1}| \leq |C_k|,$$

we have $\frac{n}{k} < |C_{k-1}| \leq |C_k|$ and so $\sum_{i=1}^{k} |C_i| > \frac{n(k-2)}{k} + \frac{2n}{k} = n$, a contradiction. This proves the lemma. □

By Brook's theorem [223], for any graph G, $\chi(G) \leq \Delta(G) + 1$, where $\chi(G)$ (the chromatic number) denotes the minimum number of colors we need in order to color the vertices of G such that no edge is monochromatic and $\Delta(G)$ denotes the maximum degree of a vertex in G. If in addition, G is not an odd cycle or a complete graph then $\chi(G) \leq \Delta(G)$ [223]. In particular, if G is of maximum degree 3 then it is 3-colorable except when $G = K_4$. This immediately gives us the following corollary.

Corollary 8.1 *Let G be a graph of maximum degree 3 on n vertices and is not a complete graph. Then G has an odd cycle transversal of size at most $n/3$.*

We obtain an improved algorithm for finding a minimum odd cycle transversal in 3-colorable graphs by applying the parameterized algorithm due to Reed, Smith and Vetta [202] which has a running time of $O(3^k kmn)$, with the parameter k running from 1 to $n/3$, the upper bound on the size of a minimum odd cycle transversal.

Theorem 8.3 *Let G be a graph on n vertices that is 3-colorable. Then an odd cycle transversal of G can be found in time $O^*(3^{n/3} = 1.4423^n)$.*

Since a maximum degree 3 graph is 3-colorable except when it is K_4, we obtain the following corollary to Theorem 8.3.

Corollary 8.2 *Let G be a graph on n vertices with maximum degree 3. Then a minimum odd cycle transversal of G can be found in time $O^*(3^{n/3} = 1.4423^n)$.*

We further improve the time complexity of finding a minimum odd cycle transversal in graphs with maximum degree 3 based on a reduction from Odd Cycle Transversal to Edge Bipartization Set in graphs with maximum degree 3. An instance of the Edge Bipartization Set problem is a graph $G = (V, E)$ and the goal is to find a minimum set of edges whose deletion makes G bipartite.

Lemma 8.3 *Let $G = (V, E)$ be a graph with maximum degree 3. Then G has an odd cycle transversal of size at most k if and only if G has an edge bipartization set of size at most k.*

Proof:

($\Longrightarrow$) Let $U \subseteq V$ be an odd cycle transversal of size k. Then $G[V - U]$ is bipartite. Let V_1 and V_2 be a bipartition of $G[V - U]$. Since G is maximum degree 3, for every vertex $u \in U$ there exists a set V_i of the bipartition $\{V_1, V_2\}$ such that u has at most one edge e incident with the vertices of V_i. Remove this edge e and include u in V_i. Repeating this for every vertex in U results in a bipartite graph G' which has at most k edges missing from G.

($\Longleftarrow$) Let $F \subseteq E$ be an edge bipartization set of G of size k. Construct a set U by adding a vertex from each of the edges in F. Clearly the size of U is at most k and $G[V - U]$ is a bipartite graph. This proves the lemma. $\square$

Note that Lemma 8.3 proves that in graphs of maximum degree 3, a minimum edge bipartization set and a minimum odd cycle transversal have the same size. Lemma 8.3 also gives us a method to obtain a minimum odd cycle transversal (U) from a minimum edge bipartization set (F). U is obtained from F by adding exactly one vertex from each of the edges in F as in the proof of Lemma 8.3. By Corollary 8.1, we also know that the size of a minimum odd cycle transversal in graphs with maximum degree 3 is at most $n/3$. We apply the parameterized algorithm for Edge Bipartization Set mentioned in Theorem 8.4, with k running from 1 to $n/3$, and obtain an improved algorithm for finding a minimum odd cycle transversal in graphs of maximum degree 3.

Theorem 8.4 [138] *Let $G = (V, E)$ be an undirected graph on n vertices and m edges. We can determine whether G has an edge bipartization set of size at most k in time $O^*(2^k)$. This algorithm takes polynomial space.*

Theorem 8.5 *Let G be a graph on n vertices that is not a complete graph. If G has maximum degree 3 then a minimum odd cycle transversal and a minimum edge bipartization set of G can be found in $O^*(2^{n/3}) = O^*(1.26^n)$ time.*

8.2.3 3- and 4-Hitting Set Problems

The Hitting Set (HS) problem is defined as follows:

Instance A finite family of sets $S_1, S_2, \ldots, S_m$ comprised of elements from a universal set U.

Goal Find a minimum sized subset $T \subseteq U$ such that $S_i \cap T \neq \emptyset$ for all i.

The 3- and 4-HS problems are special cases of the Hitting Set problem. In the 3-HS problem, $|S_i| \leq 3$ $(1 \leq i \leq m)$ and in the 4-HS problem, $|S_i| \leq 4$ $(1 \leq i \leq m)$. The parameterized versions of these problems have been shown to be fixed parameter tractable by Niedermeier et al [190]. Their result was improved by Fernau in [110], the main results of which can be summarized in the following theorem.

Theorem 8.6 [110] *The parameterized version of the* 3-*HS and the* 4-*HS problem can be solved in time* $O^*(2.179^k)$ *and* $O^*(3.115^k)$ *respectively.*

Using the values of c from Theorem 8.6, we obtain the following values of λ for the 3- and 4-HS problems: 3-HS: $c = 2.179$ and $\lambda = 0.738$; 4-HS: $c = 3.115$ and $\lambda = 0.5943$. Applying algorithm Exact() with the above values of λ and the parameterized algorithms by Fernau gives us the following theorem.

Theorem 8.7 *The* 3- *and* 4-Hitting Set *problems can be solved exactly in time* $O^*(1.7768^n)$ *and* $O^*(1.9646^n)$, *where* $n = |U|$.

Recently Wahlström [222] proposed an exact algorithm for the 3-HS problem with time complexity $O^*(1.6316^n)$. This algorithm does not directly generalize to the 4-HS problem. To the best of our knowledge, our algorithm is the first exact algorithm for the 4-HS problem with the base of the exponent less than 2.

8.2.4 Feedback Set Problems in Tournaments

The Feedback Arc (Vertex) Set problem in directed graphs is defined as follows:

Instance A directed graph $G = (V, E)$.

Goal Find a minimum sized subset $F \subseteq E$ $(F \subseteq V)$ such that $G' = (V, E - F)$ $(G' = (V - F, E'))$ is acyclic.

Problem	***Universe***	***CPA***	λ	***CEA***	***Ref.***
Cluster Vertex Deletion	Vertex Set	1.53	0.878	1.452	[132]
Cograph Vertex Deletion	Vertex Set	3.115	0.595	1.965	[132]
Constrained Minimum Vertex Cover in Bipartite Graphs	Vertex Set	1.26	0.942	1.25	[56]
Constraint Bipartite Vertex Cover	Vertex Set	1.3999	0.908	1.36	[113]
Set Splitting	Universe	2.6494	0.66	1.9	[172]
Steiner Tree	Vertex Set	$(2+\epsilon)$	0.5	1.414	[184]
Triangle Edge Deletion	Edge Set	2.179	0.738	1.7768	[110]
Triangle Vertex Deletion	Vertex Set	2.179	0.738	1.7768	[110]

Figure 8.2: Results obtained by applying Algorithm Exact to some optimization problems.

In Chapter 4 we described parameterized algorithms for the FEEDBACK ARC SET and the FEEDBACK VERTEX SET problems in tournaments with running times $O^*(2.415^k)$ and $O^*(2.27^k)$ respectively. Using these algorithms, with $c = 2.415$ we get $\lambda = 0.696$ for FEEDBACK ARC SET in tournaments and with $c = 2.27$ we get $\lambda = 0.72$ for FEEDBACK VERTEX SET in tournaments. Then using Algorithm Exact() we obtain the following theorem.

Theorem 8.8 *Let $G = (V, E)$ be a tournament with n vertices and m arcs. Then the minimum feedback arc set and feedback vertex set can be found in time $O^*(1.84821^m)$ and $O^*(1.80933^n)$ respectively.*

Observe that in any directed graph, the size of the minimum feedback arc set is at most $m/2$. This fact ensures that Algorithm Exact() will never use brute-force as $\frac{m}{2} < 0.696m$. Hence the time complexity of the algorithm is bounded by $O^*((2.415)^{m/2})$ and therefore we get the following theorem.

Theorem 8.9 *Let $G = (V, E)$ be a tournament with n vertices and m arcs. Then the minimum feedback arc set can be found in time $O^*(1.5541^m)$.*

We give several other applications of Theorem 8.1 in Figure 8.2. In Figure 8.2, *CPA* denotes the constant of the parameterized algorithm of the problem, λ is the cut value computed by the algorithm Exact() and *CEA* represents the constant of the exact algorithm obtained by applying Theorem 8.1. See reference in last column for problem definitions, and the corresponding FPT algorithms.

8.3 Conclusion

In this chapter we first developed a technique by which we can use a parameterized algorithm of time complexity $O^*((4-\epsilon)^k)$, where k is the parameter and $\epsilon > 0$, to obtain an exact algorithm of time complexity $O^*((2-\eta)^n)$, $\eta > 0$ and then applied the technique to various problems.

9 Exact Algorithms Using Enumertaion of Maximal Independent Sets

In this chapter, we illustrate the idea of designing exact algorithms by *enumerating maximal independent sets (MIS)* in a graph. Algorithms for enumerating maximal independent sets in a graph have been at the heart of many exact algorithms for the GRAPH COLORING problem [88, 41, 25]. In this chapter, we extend the power of this technique and obtain significantly improved exact algorithms for (1) ODD CYCLE TRANSVERSAL, (2) MAXIMUM k-COLORABLE INDUCED SUBGRAPH, (3) MINIMUM MAXIMAL MATCHING, (4) MINIMUM EDGE DOMINATING SET and (5) MATRIX DOMINATING SET. Though all our algorithms use the subroutine for enumerating maximal independent sets, we transform the problems or use interesting structural characterizations to make use of the MIS enumeration algorithm.

This chapter is organized as follows. In the next section we list out known results for enumeration of maximal independent sets, to find a maximum independent set and to find the chromatic number of a graph. Then Section 9.2, 9.3, 9.4 and 9.4.2 apply the results of Section 9.1 to obtain efficient exact algorithms for minimum odd cycle transversal, maximum k-colorable induced subgraph, minimum maximal matching and minimum edge dominating set respectively. Section 9.4.2 also contains an application of minimum edge dominating set for matrix domination.

9.1 Preliminaries

The first two theorems below provide bounds on the total number of maximal independent sets of size k and the third bounds the total number of maximal independent sets in a graph.

Theorem 9.1 [89] *The number of maximal independent sets of size k in a graph is at most $3^{4k-n}4^{n-3k}$.*

Theorem 9.2 [41] *Let $G = (V, E)$ be a graph on n vertices. The maximum number of maximal independent sets of size k in G is*

$$N(n, k) = \lfloor n/k \rfloor^{(\lfloor n/k \rfloor+1)k-n}(\lfloor n/k \rfloor + 1)^{n-\lfloor n/k \rfloor k}.$$

In fact, all maximal independent sets of size at most k can be listed in time proportional to $N(n, k)$.

Theorem 9.3 [154, 185] *Let $G = (V, E)$ be a graph on n vertices, then G contains at most $3^{n/3} = (1.44225)^n$ maximal independent sets. All the maximal independent sets can be enumerated in $O^*(1.44225^n)$ time.*

We also need the following theorem.

Theorem 9.4 [116],[208],[209] *Given a graph $G = (V, E)$ on n vertices, a* Maximum Independent Set *can be found in*

1. *$\mathcal{O}(1.2210^n)$ time and space polynomial in n or*
2. *$\mathcal{O}(1.2108^n)$ time $\big($ or $\mathcal{O}(2^{n/4})\big)$ time and space exponential in n.*

9.2 Minimum Odd Cycle Transversal

The Odd Cycle Transversal problem is defined below:

> Odd Cycle Transversal (OCT): Given a graph $G = (V, E)$, find a subset $O \subseteq V$ of minimum size such that $G[V - O]$ is bipartite.

Byskov [40] gave an algorithm that enumerates all minimal OCT's in a graph with running time $O^*(1.7724^n)$. We give an improved algorithm to find a minimum sized OCT which can be modified to count all minimum sized OCT's. We first give a characterization of a minimum odd cycle transversal (OCT) in terms of maximal independent sets and then use it to obtain an improved exact algorithm.

Theorem 9.5 *Let $G = (V, E)$ be a connected undirected graph and let O be a minimum odd cycle transversal of G. Then $V - O$ can be partitioned into V_1 and V_2 such that V_1 is a maximal independent set of G and V_2 is a maximum independent set of $G[V - V_1]$.*

Proof: Since O is an odd cycle transversal of G, $G[V - O]$ is bipartite. Let K and L be a bipartition of $G[V - O]$. Add isolated vertices (if any) from either K to L or from L to K such that one of them, say K, becomes a maximal independent set of $G - O$. Note that since O is a minimum odd cycle transversal of G, for every vertex $x \in O$ there exists a witness odd cycle C_x such that $C_x \cap O = \{x\}$. Hence every vertex $x \in O$ has at least one neighbor in K and one in L. This shows that K is also maximal in G. Set $V_1 = K$ and $V_2 = L$. It remains to show that V_2 is a maximum independent set of $G[V - V_1]$. Suppose not. Let W be a maximum independent set of $G[V - V_1]$. Then $|W| > |V_2|$. Observe that $(V - V_1 - W)$ is an odd cycle transversal of G. Now

$$\begin{aligned} |V - V_1 - W| &= |V| - |V_1| - |W| && (\text{since } V_1 \cap W = \emptyset) \\ &= |O| + |V_2| - |W| && (\text{since } |V| = |V_1| + |V_2| + |O|) \\ &< |O| && (\text{since } |V_2| < |W|) \end{aligned}$$

This contradicts the fact that O is a minimum odd cycle transversal of G. □

Observe that any other *maximum* independent set of $G - V_1$, say T, also gives us a minimum odd cycle transversal. This is because $(V - V_1 - T)$ is an OCT of size $|V - V_1 - T| = |V - V_1 - V_2|$. This observation is the basis of the algorithm in Figure 9.1.

The correctness of the algorithm follows from Theorem 9.5. Theorems 9.3 and 9.4 give, respectively, a bound on the total number of maximal independent sets enumerated in Step 2 and the time taken to find a maximum independent set in

```
Algorithm Min OCT($G$)
Input: A connected undirected graph $G = (V, E)$.
Output: A minimum odd cycle transversal of $G$.

Step 1 Set $X \leftarrow V$ ($X$ stores a minimum odd cycle transversal.)
Step 2 Enumerate all maximal independent sets of $G$, and for each
    MIS $I$,
    Step 3a Find a maximum independent set of $G - I$, say $I'$.
    Step 3b If $|X| > |V - I - I'|$ then set $X = V - I - I'$.
Step 4 Return $X$.
```

Figure 9.1: Algorithm for finding a minimum odd cycle transversal of a graph.

Step 3*a*. Hence, the time complexity of the algorithm is upper bounded by

$$\sum_{k=1}^{n} |MIS_k| 2^{(n-k)/4} \;\leq\; 3^{n/3} 2^{n/4} \;\leq\; 1.7152^n, \tag{9.1}$$

where MIS_k represents the set of all maximal independent sets of size k. Thus this algorithm takes $O^*(1.7152^n)$ which improves the $O^*(1.7724^n)$ time algorithm of Byskov [40].

We next present a refined time analysis of the algorithm using tighter bounds on the number of maximal independent sets of small size.

9.2.1 A Refined Timing Analysis

We break up the summation in equation (9.1) as follows

$$\sum_{k=1}^{n} |MIS_k| 2^{(n-k)/4} = \sum_{k=1}^{\lfloor 0.33n \rfloor} |MIS_k| 2^{(n-k)/4} + \sum_{k=\lfloor 0.33n \rfloor+1}^{n} |MIS_k| 2^{(n-k)/4} \tag{9.2}$$

We will bound each term on the right side of (9.2) individually. To bound the first term, we use the bound in Theorem 9.1 on the number of maximal independent sets of size k: $N(k) \leq 3^{4k-n} 4^{n-3k}$ for $1 \leq k \leq \lfloor 0.33n \rfloor$. It is easy to verify that

$N(k)$ is an increasing function of k. Thus,

$$\begin{aligned}
\sum_{k=1}^{\lfloor 0.33n \rfloor} |MIS_k| 2^{(n-k)/4} &\leq \sum_{k=1}^{\lfloor 0.33n \rfloor} 3^{4k-n} 4^{n-3k} 2^{(n-k)/4} \\
&\leq \left(\frac{4.2^{1/4}}{3}\right)^n \cdot \sum_{k=1}^{0.33n} \left(\frac{3^4}{4^3.(2^{1/4})}\right)^k \\
&\leq (0.33n)(1.58561)^n \left(\frac{3^4}{4^3.(2^{1/4})}\right)^{0.33n} \quad \leq O^*(1.62^n)
\end{aligned}$$

To bound the second term, we use the bound on the total number of maximal independent sets in a graph, which is $3^{n/3}$. This gives,

$$\begin{aligned}
\sum_{k=\lfloor 0.33n \rfloor + 1}^{n} |MIS_k| 2^{(n-k)/4} &\leq 3^{n/3} \sum_{k=\lfloor 0.33n \rfloor + 1}^{n} 2^{(n-k)/4} \\
&\leq (0.67n) 3^{n/3} 2^{(0.67n)/4} \leq O^*(1.62^n)
\end{aligned}$$

This gives us following theorem.

Theorem 9.6 *Let $G = (V, E)$ be an undirected graph with n vertices. A minimum odd cycle transversal of G can be found in $O^*(1.62^n)$ time.*

9.2.2 Counting all Minimum Odd Cycle Transversals

The algorithm for ODD CYCLE TRANSVERSAL presented in the previous section can be generalized to the *counting version* of the problem (#OCT) where the objective is to count all minimum sized odd cycle transversals. To do this, the only modification we need is to use an algorithm to *count* all maximum independent sets in Step 3a of the minimum odd cycle transversal algorithm in Figure 9.1. Using the $O^*(1.2461^n)$ time algorithm developed by Fürer and Kasiviswanathan [123] to count all maximum independent sets in a graph, we obtain the following theorem

Theorem 9.7 *Let $G = (V, E)$ be an undirected graph on n vertices. Then we can count the number of minimum odd cycle transversals in $O^*(1.6713^n)$ time.*

9.3 Maximum k-Colorable Induced Subgraph

We first recall the definition of proper coloring and k-colorable graphs. A *proper coloring* of a graph is an assignment of colors to its vertices such that no edge is monochromatic, that is end points of every edge get different colors. A graph is called *k-colorable* if there exists a proper coloring with at most k colors. In the last section we gave an exact algorithm to find a minimum Odd Cycle Transversal, which can be seen as a special case of Maximum k-Colorable Induced Subgraph (M-k-CIS). M-k-CIS is a problem of finding a maximum subset of vertices such that the subgraph induced on these vertices is k-colorable. In fact in this section we go a step further and study a further generalization of M-k-CIS problem. To define the generalized problem, we first define kl-graphs. A graph $G = (V, E)$ is called a kl-graph if V can be can be partitioned into k independent sets and l cliques. Now the generalized problem is defines as follows.

> Maximum kl-Induced Subgraph (M-kl-IS): Given a graph $G = (V, E)$ and positive integers k and l, find a maximum size subset $V' \subseteq V$ such that $G[V']$ is a kl-graph.

We also assume that $k + l \leq n$, as otherwise there does not exist a kl-induced subgraph. The M-k-CIS problem is equivalent of finding a maximum subset of vertices such that the graph induced on this subset can be partitioned into k independent sets. Hence when $l = 0$, M-kl-IS problem corresponds to M-k-CIS and when $k = 0$, it corresponds to the problem of M-l-CIS in the edge complement graph of the input graph. An *edge complement graph* of $G = (V, E)$ is a graph on the same set of vertices but there is an edge between $u \in V$ and $v \in V$ if and only if they are non adjacent vertices in G, that is $(u, v) \notin E$. We denote the edge complement graph of G by $\overline{G}$. For $k = 1$ and $l = 1$, M-kl-IS corresponds to the problem of finding a maximum vertex induced split subgraph (a graph in which the vertices can be partitioned into a clique and an independent set).

Though M-kl-IS is a vertex subset problem, we can not solve the problem by just enumerating all subsets of the vertex set of a graph, as testing whether the input graph is a kl-graph is a NP-complete problem if either of k or l is more than 2 [39]. Brandstädt [39] gave a polynomial time algorithm to test whether the input graph can be partitioned into at most 2 independent sets or 2 cliques, that is testing whether the input graph is a kl-graph for $0 \leq k, l \leq 2$.

Now we give a simple lemma which will be useful in the proof of the main result later.

Lemma 9.1 *If there is a subset $S \subseteq V$, $|S| = t \geq k + l$, such that it is an induced subgraph on $k' \leq k$ independent sets and $l' \leq l$ cliques then there exists an kl-induced subgraph on t vertices.*

Proof: Let S be partitioned as $I_1, \cdots, I_{k'}$ independent sets and $C_1, \cdots, C_{l'}$ cliques. Our proof relies on the fact that every singleton vertex is an independent set as well as a clique. We select and delete a set X of $p = (k - k') + (l - l')$ vertices from $I_1, \cdots, I_{k'}, C_1, \cdots, C_{l'}$ in such a way that none of these sets becomes empty. We can select such an X because $t \geq k + l \geq k' + l'$. Now we treat every vertex in $x \in X$ as a set in itself which is a clique as well as an independent set. Thus $I_1, \cdots, I_{k'}, C_1, \cdots, C_{l'}, \{x\}$, $\forall x \in X$, form a kl-induced subgraph on t vertices. $\square$

Our algorithm for finding a maximum kl-induced subgraph is based on a reduction to finding a maximum independent set in an auxiliary graph. We now give the construction of this auxiliary graph.

Construction : Given a graph $G = (V, E)$ and positive integers k and l, construct $H^{kl} = (V^{kl}, E^{kl})$ as follows. Take $k + l$ copies of V. That is $V_i = \{u_i \mid u \in V\}$ for $1 \leq i \leq k + l$. So $V^{kl} = \cup_{i=1}^{k+l} V_i$. Now if there is an edge $(u, v) \in E$ then edges of the form (u_i, v_i) for $1 \leq i \leq k$ are in E^{kl} and if there is no edge of the form $(u, v) \in E$ then edges of the form (u_i, v_i) for $k + 1 \leq i \leq k + l$ are in E^{kl}, that is $H^{kl}[V_i]$ is a copy of G for $1 \leq i \leq k$ and $H^{kl}[V_i]$ is a copy of $\overline{G}$ for $k + 1 \leq i \leq l$. Now take the set $A_u = \{u_i \mid u_i \in V_i,\ \ 1 \leq i \leq k + l\}$, for $u \in V$. Now for every pair of vertices in A_u add an edge in H^{kl}. That is $H^{kl}[A_u]$ is a complete graph. This completes the construction.

Now we show the following theorem which relates a kl-induced subgraph of a graph G to an independent set of the associated auxiliary graph H^{kl}.

Theorem 9.8 *Let $G = (V, E)$ be a graph on n vertices and k and l be positive integers such that $k + l \leq n$. Then G has a kl-induced subgraph of size t if and only if H^{kl} has an independent set of size t.*

Proof: Let $S \subseteq V$ be such that $G[S]$ is a kl-induced subgraph of G of size t. Let S be partitioned as $S_1, S_2 \cdots, S_k, S_{k+1}, \cdots, S_l$. S_i forms an independent set for $1 \leq i \leq k$ and a clique when $k+1 \leq i \leq l$ in $G[S]$. We also know that $H^{kl} = (V^{kl}, E^{kl})$ where $V^{kl} = \cup_{i=1}^{k+l} V_i$. Take the image of S_i in V_i that is let S_i' be the set of vertices corresponding to the vertices in S_i in the vertex set V_i. Then we claim that

$$I = \bigcup_{i=1}^{k+l} (V_i \cap \{u_i \mid u \in S_i\}) = \bigcup_{i=1}^{k+l} S_i'$$

is an independent set of size t in H^{kl}. Towards this end it is enough to observe that S_i's partition S, $H^{kl}[V_i]$ is a copy of G for $1 \leq i \leq k$ and a copy of edge complement of G, that is $\overline{G}$, for $k+1 \leq i \leq l$ and the only edges between any pair of copies of G or $\overline{G}$ in H^{kl}, say $H^{kl}[V_i]$ and $H^{kl}[V_j]$, are of the form $(u_i, u_j), u \in V$.

Conversely let K be an independent set of H^{kl} of size t and K be decomposed as K_i, $1 \leq i \leq k+l$, where $K_i = K \cap V_i$. Let $B_i' = \{u \mid u \in V,\ u_i \in K_i\}$ for $1 \leq i \leq k+l$. Observe that for any $u \in V$, K contains at most one of the copies of u, i.e. $|K \cap \{u_1, u_2 \cdots, u_{k+l}\}|$ is either 0 or 1 for $u \in V$. For any pair of indices $1 \leq i < j \leq k+l$, $B_i' \cap B_j' = \emptyset$. Thus $| \cup_{i=1}^{k+l} B_i| = \sum_{i=1}^{k+l} |B_i'| = t$. The adjacency's in $H^{kl}[V_i]$ for $1 \leq i \leq k$ is the same as G and hence B_i''s are independent sets in G. For $k+1 \leq i \leq l$, the adjacency's in $H^{kl}[V_i]$ is the same as $\overline{G}$ and hence B_i''s are cliques in G. But notice that some of these B_i' could be empty. So $G[\cup_{i=1}^{k+l} B_i']$ is a $k'l'$-induced subgraph on t vertices where $k' \leq k$ and $l' \leq l$ are the number of non empty B_i', $1 \leq i \leq k$ and $k+1 \leq i \leq k+l$ respectively. Now we apply Lemma 9.1 and make $G[\cup_{i=1}^{k+l} B_i']$ a kl-induced subgraph of size t. This completes the proof. □

So to obtain a maximum kl-induced subgraph of a graph G on n vertices we need to find a maximum independent set of H^{kl} which has $(k+l)n$ vertices. Hence as an immediate consequence to the Theorem 9.8, we have the following.

Theorem 9.9 *Let $G = (V, E)$ be a graph on n vertices then M-kl-IS problem can be solved in $\mathcal{O}(\alpha_{mis}^{(k+l)n})$, where α_{mis} is the base of the running time of the fastest known algorithm finding a maximum independent set, which is* 1.2108.

Byskov [40, 41] considered the problem of enumerating maximal k-colorable induced subgraphs and showed that all maximal 3-colorable subgraph and in general k-colorable subgraph for $k \geq 4$ can be enumerated in time $\mathcal{O}(2.1809^n)$ and

$\mathcal{O}(2.4023^n)$ respectively. But even when we restrict ourselves to finding *a maximum* sized 3-colorable or 4-colorable induced subgraph of G, these algorithms do not appear to give better time than for an arbitrary k, as the time complexity is dominated by the dynamic programming over subsets of the vertex set of the graph. As a corollary to Theorem 9.9 with α_{mis} from the Theorem 9.4, we obtain this.

Corollary 9.1 *Let G be a graph on n vertices. Then M-k-CIS problem can be solved in $\mathcal{O}(1.4908^n)$, $\mathcal{O}(1.8203^n)$ and $\mathcal{O}(2.2226^n)$ time and space polynomial in n for $k = 2, 3$ and 4 respectively. If we are willing to use exponential space then the running time for M-k-CIS problem can be improved to $\mathcal{O}(1.4660^n)$, $\mathcal{O}(1.7751^n)$ and $\mathcal{O}(2.21493^n)$ for $k = 2, 3$ and 4 respectively.*

The time complexity of $\mathcal{O}(1.4908^n)$ (or $\mathcal{O}(1.4660^n)$) for M-2-CIS problem improves the $\mathcal{O}(1.62^n)$ time algorithm presented in the last section for the equivalent problem of finding an Odd Cycle Transversal or a Maximum Bipartite Subgraph of a given graph. Another corollary to Theorem 9.9 includes an algorithm to find a maximum induced split subgraph.

Corollary 9.2 *Let G be a graph on n vertices. Then M-11-IS (maximum induced split graph) problem can be solved in time $\mathcal{O}(1.4908^n)$ and space polynomial in n or in time $\mathcal{O}(1.4660^n)$ and space exponential in n.*

The time complexity for M-4-CIS problem can be slightly improved using a different argument based on the following lemma.

Lemma 9.2 *Let $G = (V, E)$ be a graph on n vertices and S be a maximum k-colorable induced subgraph of G. Then there exists a partition of S as k color classes, namely $S_1, S_2, \ldots, S_k$, such that S_1 is a maximal independent set of G and $G[\cup_{i=2}^{k} S_i]$ is a maximum $k-1$ colorable induced subgraph of $G[V - S_1]$.*

Proof: Let S be a subset of V such that $G[S]$ is a maximum k colorable induced subgraph of G and $S_1, S_2, \cdots S_k$ be its k color classes. Make S_1 a maximal independent set of G by adding vertices from other color classes if possible. Now it only remains to note that $G[\cup_{i=2}^{k} S_i]$ is a maximum $k-1$ colorable subgraph of $G[V - S_1]$, otherwise it would contradict the fact that S is a maximum k-colorable induced subgraph of G. $\square$

Now to obtain an improved algorithm for M-4-CIS problem we first enumerate all maximal independent sets using Theorems 9.2 and 9.3 and then apply the algorithm for M-3-CIS problem listed in Corollary 9.1. Let β be the base of the runtime for M-3-CIS problem listed in Corollary 9.1 and MIS_k be the set of maximal independent sets of size k. So the time complexity of the algorithm is dominated by the following.

$$\begin{aligned}
\sum_{k=1}^{n} |MIS_k|\beta^{(n-k)} &= \sum_{k=1}^{\lfloor 0.33n \rfloor} |MIS_k|\beta^{(n-k)} + \sum_{k=\lfloor 0.33n \rfloor+1}^{n} |MIS_k|\beta^{(n-k)} \\
&\leq \sum_{k=1}^{\lfloor 0.33n \rfloor} 3^{4k-n}4^{n-3k}\beta^{(n-k)} + 3^{n/3} \sum_{k=\lfloor 0.33n \rfloor} \beta^{(n-k)} \\
&\leq n\left(\left(\frac{4.\beta}{3}\right)^{n} \cdot \left(\frac{3^4}{4^3.\beta}\right)^{0.33n} + 3^{n/3}\beta^{(0.67n)}\right) \qquad (9.3)
\end{aligned}$$

To bound the first term, we use the bound in Theorem 9.2 on the number of maximal independent sets of size k and to bound the second term, we use the bound in Theorem 9.3 on the total number of maximal independent sets in a graph, which is $3^{n/3}$. Using $\beta = 1.8203$ or 1.7741 in Equation 9.3, we obtain the following theorem.

Theorem 9.10 *Let $G = (V, E)$ be a graph on n vertices. Then M-4-CIS problem can be solved in $\mathcal{O}(2.1528^n)$ time and space polynomial in n or in time $\mathcal{O}(2.1168^n)$ and space exponential in n.*

9.4 Minimum Maximal Matching

A Minimum Maximal Matching (abbreviated MMM) of a graph $G = (V, E)$ is a maximal matching of minimum cardinality. In this section, we first give an algorithm which finds a minimum maximal matching of G in $O^*(2^n)$ time and then improve this to obtain an $O^*(1.44225^n)$ time algorithm. This improves an $O^*(1.4422^m)$ time algorithm of Randerath and Schiermeyer [200] for the problem. We need a lemma.

Lemma 9.3 *Given a graph $G = (V, E)$, let M be a maximal matching of G. Let*

$U = \{v \in V : v$ is an end point of some edge of $M\}$. If M' is any maximum matching of $G[U]$ then M' is a maximal matching of G and $|M'| = |M|$.

Proof: Since M is a perfect matching of $G[U]$, it is a matching of maximum size in $G[U]$, and so $|M'| = |M|$. To see that M' is maximal in G, observe that U is a vertex cover of G and that the vertex set of M' is precisely U. Thus M' cannot be augmented any further proving that it is maximal. □

Corollary 9.3 *Let $G = (V, E)$ be a graph and M be a minimum maximal matching of G. Let U be a subset of vertices containing the endpoints of M. If M' is any maximum matching of $G[U]$ then M' is a minimum maximal matching of G.*

Thus to find a minimum maximal matching of G, we enumerate all even-cardinality subsets $U \subseteq V$, find a maximum matching M of $G[U]$, check for M's maximality in G and finally output the one with minimum size. Note that at any point we store at most two subsets of V, so our algorithm takes polynomial space. This gives us the following theorem.

Theorem 9.11 *Let $G = (V, E)$ be a graph on n vertices. We can find a* Minimum Maximal Matching *of G in $O^*(2^n)$ time and space polynomial in n.*

9.4.1 Improved Algorithm

We provide a characterization of minimum maximal matchings in terms of minimal vertex covers (complement of maximal independent sets) which is used to obtain an improved algorithm.

Theorem 9.12 *Given any graph $G = (V, E)$, there exists a minimal vertex cover U of G such that if*

1. *M_1 is a maximum matching of $G[U]$, and*

2. *M_2 is a maximum matching of $G[V - R]$, where R is the set of endpoints of M_1,*

then $M_1 \cup M_2$ is a minimum maximal matching of G.

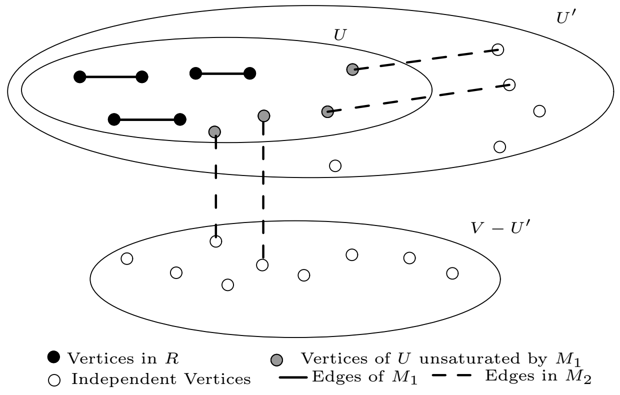

Figure 9.2: Characterization of minimum maximal matching.

Proof: Let M be a minimum maximal matching of G and let U' be the set of endpoints of M. Since M is a maximal matching of G, U' is a vertex cover of G and hence contains a minimal vertex cover $U \subseteq U'$ of G. See Figure 9.2. Let M_1 and M_2 be as stated in the theorem. Note that M_1 and M_2 are disjoint and that $N = M_1 \cup M_2$ is a maximal matching of G. We will show that N is actually a minimum maximal matching by showing that $|N| \leq |M|$.

Since U is a vertex cover of G, every edge of M either has both its endpoints in U or has one endpoint in U and the other in $V-U$. Let K be the set of edges of M with both endpoints in U and L the set of edges of M with one endpoint in U and the other in $V-U$. K is a matching of $G[U]$ and since M_1 is a matching of $G[U]$ of maximum size, $|K| \leq |M_1|$. Recall that U' is the set of endpoints of M. Since $U \subseteq U'$, every vertex of U is an endpoint of an edge in either K or in L. Therefore, $|L| = |U| - 2|K|$. A similar argument shows that $2|M_1| + |M_2| \leq |U|$. Suppose $|M_1| = |K| + r$, $r \geq 0$. Then,

$$\begin{aligned}
|N| = |M_1| + |M_2| &\leq |M_1| + (|U| - 2|M_1|) \\
&= |K| + r + |U| - 2(|K| + r) \\
&= |K| + (|U| - 2|K|) - r \\
&= |K| + |L| - r && \text{(Since, } |L| = |U| - 2|K|) \\
&\leq |M| && \text{(since } r \geq 0).
\end{aligned}$$

This proves the theorem. □

We use this characterization to get an algorithm to find a minimum maximal matching of G. The algorithm is given in Figure 9.3. The correctness of the

```
Algorithm MMM($G$)
Input: A connected undirected graph $G$.
Output: A minimum maximal matching of $G$.

Step 0 Find a maximal matching $P$ of $G$
Step 1 Set $X \leftarrow P$ ($X$ stores a minimum maximal matching of $G$.)
Step 2 Enumerate all maximal independent sets of $G$, and for each
    MIS $I$,
    Step 3a Find a maximum matching of $G[V - I]$, say $M'$.
    Step 3b Find a maximum matching $M''$ of $G[V - R]$, where
        $R$ is the set of vertices containing the end points of $M'$.
    Step 3c If $|X| > |M \cup M''|$ then set $X = M \cup M''$.
Step 4 Return $X$.
```

Figure 9.3: Algorithm for finding a minimum maximal matching of a graph.

algorithm follows from Theorem 9.12 and the fact that complement of a maximal independent set is a minimal vertex cover. The time complexity of the algorithm is dominated by the enumeration of maximal independent sets which by Theorem 9.3 can be done in $O^*(1.44225^n)$ time. This gives us following theorem:

Theorem 9.13 *Let $G = (V, E)$ be a graph on n vertices. We can find a* Minimum Maximal Matching *of G in $O^*(1.44225^n)$ time.*

9.4.2 Minimum Edge Dominating Set

A minimum edge dominating set (MEDS), D, of a graph $G = (V, E)$ is a set of edges of minimum size such that every edge of G is either in D or is adjacent to some edge in D. An algorithm of time complexity $O^*(1.4422^m)$ appears in [200]. Our algorithm depends on a classical result which shows that every minimum maximal matching is a MEDS [124]. We give a proof for completion.

Lemma 9.4 [124] *Let $G = (V, E)$ be a graph. Then every minimum maximal matching of G is a minimum edge dominating set.*

Proof: Every minimum maximal matching is an edge dominating set. So,

$$|MEDS| \leq |MMM|.$$

To show the desired result, we will show that there exists a MEDS which is also a maximal matching.

Let F be a MEDS. Consider the edge induced sub graph H on F. Any path in H has length at most 2. This is because if $P = xyzw\dots$ is a path of length ≥ 3 then $F - \{(y, z)\}$ is an edge dominating set of smaller size. So the connected components of H are stars (a star is a tree consisting of one vertex adjacent to all others) and every vertex other than the central vertex has an edge incident on it which is not part of F (because of minimality of F).

Assume to the contrary that F is not a maximal matching. Let F be a MEDS such that the edge induced subgraph H on F has maximum number of isolated edges as its connected components. Now take the star $K_{1,s}$, $s \geq 2$ with v as its central vertex and u as its one of neighbors then there exists an edge $e = (u, w)$ such that e is not part of F because of minimality of F. Now take $F' = F - \{(v, u)\} + \{(u, w)\}$. Clearly F' is a MEDS and has more disjoint edges in it than F. This proves the lemma. □

Lemma 9.4 and Theorem 9.13 give us the following theorem.

Theorem 9.14 *Given a graph G on n vertices, the* Minimum Edge Dominating Set *problem on G can be solved in time $O^*(1.44225^n)$.*

9.4.3 Matrix Domination Set

Problem:	Matrix Domination (MMD)
Instance:	An $m \times n$ matrix M with entries from $\{0, 1\}$.
Question:	Find a minimum set of non-zero entries in M that dominates all others, i.e., a minimum size subset $C \subseteq \{1, 2 \dots m\} \times \{1, 2 \dots n\}$ such that $M_{i,j} = 1$ for all $(i, j) \in C$ and such that, whenever $M_{i,j} = 1$, there exists an $(i', j') \in C$ for which either $i = i'$ or $j = j'$.

It is well known that this problem can be reduced to finding a minimum edge dominating set in a bipartite graph [124]. Given an $m \times n$ matrix M construct a bipartite graph $B = (U \uplus V, E)$ with $U = \{u_1, u_2, \cdots, u_m\}$ and $V = \{v_1, v_2, \cdots, v_n\}$ such that there is an edge between u_i and v_j if and only if $M[i, j] = 1$. Then we can easily show that a minimum edge dominating set of B is a minimum matrix dominator of M and vice versa.

Observe that the run time of the algorithms in Theorems 9.13 and 9.14 for the Minimum Maximal Matching and the Minimum Edge Dominating Set problems are upper bounded by the *total number of maximal independent sets* in a graph. For triangle-free graphs, there are better bounds known for the total number of maximal independent sets. For such graphs we know the following

Theorem 9.15 [147] *A triangle-free graph on n vertices contains at most $2^{n/2} = 1.4142^n$ maximal independent sets. All maximal independent sets can be enumerated in $O^*(1.4142^n)$ time.*

The reduction from the Matrix Domination Set to the Minimum Edge Dominating Set, transforms an $m \times n$ matrix M into a bipartite (and hence triangle-free) graph B. Theorem 9.15 then gives us

Theorem 9.16 *Let M be an $m \times n$ matrix. A minimum size matrix dominator for M can be found in time $O^*(1.4142^{n+m})$.*

9.5 Conclusion

Algorithms for enumerating maximal independent sets have been used in most of the algorithms developed for the Coloring problem. Here, we further showed the power of this technique by obtaining significantly improved and the best known exact algorithms for Minimum Odd Cycle Transversal, Maximum k-colorable Induced Subgraph for small values of k, Maximum Split Graphs, Minimum Maximal Matching and Minimum Edge Dominating Set. Though all our algorithms have enumeration of maximal independent set as a subroutine, we had to transform the problem or use interesting characterizations in order to apply the MIS enumeration algorithm.

10 Exact Algorithms Using Combination of Branching and Treewidth

One of the major techniques for constructing fast exponential time algorithms is the Branch & Reduce paradigm. Branch & Reduce algorithms (also called search tree algorithms,or Davis-Putnam-style exponential-time backtracking algorithms) solve NP hard combinatorial problems using reduction rules and branching rules. Such an algorithm is applied to a problem instance by recursively calling itself on smaller instances of the problem.

Many problems on graphs with n vertices and treewidth at most ℓ can be solved in time $\mathcal{O}(c^{\ell} n^{O(1)})$, where c is some problem dependent constant. This observation combined with upper bounds on treewidth was used to obtain fast exponential algorithms for NP hard problems on cubic, sparse and planar graphs [117, 121, 165]. For example, a maximum independent set of a graph, given with its tree decomposition of width at most ℓ, can be found in time $\mathcal{O}(2^{\ell} n)$ (see for example [28]). So, a quite natural approach to solve the independent set problem would be to branch on vertices of high degree and if a subproblem with all vertices of small degrees is obtained, then use dynamic programming over graphs of bounded treewidth. Unfortunately, such a simple approach still provides poor running time mainly because the best known upper bounds on treewidth of graphs with small maximum degree are too large to be useful.

In this chapter we show two different approaches based on combinations of branching and treewidth techniques. Both approaches are based on a careful balancing of these two techniques. In the first approach the algorithm either performs fast branching, or if there is an obstacle for fast branching, this obstacle is used for the construction of a path decomposition of small width for the original graph. We call this technique *Branching & Global Application of Width Parameters* and exemplify it on the following problems.

- Minimum Maximal Matching (MMM): Given a graph G, find a maximal matching of minimum size.
- #3-Coloring: Given a graph G, count the number of 3-colorings of G.

We give an $\mathcal{O}(1.4082^n)$ time algorithm for MMM improving on the $\mathcal{O}(1.4422^n)$ bound developed in the last chapter.

#3-Coloring is a special case of the more general $(d, 2)$-Constraint Satisfaction Problem ($(d, 2)$-CSP). A systematic study of exact algorithms for $(d, 2)$-CSP was initiated in [12] where an algorithm with running time $\mathcal{O}(1.788^n)$ was given for #3-Coloring. In recent years, the algorithms for $(d, 2)$-CSP have been significantly improved. Notable contributions include papers by Williams [225] and Fürer and Kashiviswanathan [123]. The current fastest algorithm for #3-Coloring has running time $\mathcal{O}(1.770^n)$ [123]. Here we improve this algorithm with our technique of combining branching and treewidth and give an $\mathcal{O}(1.6308^n)$ time algorithm for the problem.

In the second approach the branching occurs until the algorithm reaches a subproblem with a small number of edges (and here the right choice of the size of subproblems is crucial) and then dynamic programming on bounded treewidth or pathwidth is applied on these subproblems. We term this technique *Branching & Local Application of Width Parameters* and exemplify it on the following parameterized problem.

- k-Weighted Vertex Cover (k-WVC): Given a graph $G = (V, E)$, a weight function $w : V \to \mathbb{R}^+$ such that for every vertex v, $w(v) \geq 1$ and $k \in \mathbb{R}^+$, find if there exists a vertex cover of weight at most k, where the weight of a vertex cover C is $w(C) = \sum_{v \in C} w(v)$.

The k-Vertex Cover problem, asking whether an input graph has at most k vertices that are incident to all its edges, is a celebrated example of a FPT problem. When parameterized by k, this problem can be solved in time $\mathcal{O}(n^3 + c^k)$, where

c is a constant. After a long race of improvements (see for example [58, 191]), the current best algorithm by Chandran and Grandoni has running time $\mathcal{O}(1.2745^k k^4 + kn)$ [49].

For the k-Weighted Vertex Cover, also known as Real Vertex Cover, Niedermeier and Rossmanith [192] gave two algorithms, one with running time $\mathcal{O}(1.3954^k + kn)$ and polynomial space and the other one using time $\mathcal{O}(1.3788^k + kn)$ and space $\mathcal{O}(1.3630^k n^{O(1)})$. Their paper on k-Weighted Vertex Cover is based on branching, kernelization and the idea of "memorization". Their analysis involves extensive case distinctions when the maximum degree of the reduced graph becomes 3. Here, we give a very simple algorithm running in time $\mathcal{O}(1.3803^k n)$ and space $\mathcal{O}(1.2599^k + kn)$. The other problems for which we give parameterized algorithms in this chapter include parameterized edge dominating set and its variants.

We use standard dynamic programming algorithms on graphs of bounded pathwidth or treewidth in all of our algorithms. But to use these algorithms it is important that we have good upper bounds on the pathwidth of the subgraphs arising in our recursive algorithms. We prove several upper bounds on the pathwidth of sparse graphs that we use in our algorithms. These bounds are of independent interest.

The rest of the chapter is organized as follows. In the next section we give some basic definitions and notations we use in the chapter. We develop some non trivial upper bounds on pathwidth of sparse graphs in Section 10.2. In Section 10.3, we exemplify the technique of *branching & global application of width parameters* on MMM and #3-Coloring. In Section 10.4 we give an algorithm for k-WVC as an application of *branching & local application of width parameters* technique. In Section 10.5 we give improved parameterized algorithm for parameterized version of edge dominating set and several of its variants. Finally, we conclude with some remarks and open problems in Section 10.6.

10.1 Preliminaries

In this chapter we consider simple undirected graphs. Let $G = (V, E)$ be a graph and let n denote the number of vertices and m the number of edges of G. We denote by $\Delta(G)$ the maximum vertex degree in G. A subset of vertices $S \subseteq V$ is a *vertex cover* in G if for every edge e of G at least one endpoint of e is in S. We denote *treewidth* and *pathwidth* of a graph G by $\mathbf{tw}(G)$ and $\mathbf{pw}(G)$ respectively.

10.2 Upper Bounds on Pathwidth in Sparse Graphs

In this section we develop several upper bounds on the pathwidth of sparse graph. We need the following known bound on the pathwidth of graphs with maximum degree 3 to prove the two lemmas of this section.

Proposition 10.1 ([121]) *For any $\varepsilon > 0$, there exists an integer n_ε such that for every graph G with $n > n_\varepsilon$ vertices and maximum degree at most 3, $\mathbf{pw}(G) \leq (1/6 + \varepsilon)n$. Moreover, a path decomposition of the corresponding width can be constructed in polynomial time.*

Using Proposition 10.1 we prove the following bound for general graphs.

Lemma 10.1 *For any $\varepsilon > 0$, there exists an integer n_ε such that for every graph G with $n > n_\varepsilon$ vertices,*

$$\mathbf{pw}(G) \leq \frac{1}{6}n_3 + \frac{1}{3}n_4 + \frac{13}{30}n_5 + \frac{23}{45}n_6 + n_{\geq 7} + \varepsilon n,$$

where n_i is the number of vertices of degree i in G for any $i \in \{3, \ldots, 6\}$ and $n_{\geq 7}$ is the number of vertices of degree at least 7. Moreover, a path decomposition of the corresponding width can be constructed in polynomial time.

Proof: Let $G = (V, E)$ be a graph on n vertices. It is well known (see for example [31]) that if the treewidth of a graph is at least 2, then contracting edges incident on vertices of degree 1 and 2 does not change the treewidth of a graph. But after contracting all the edges incident on vertices of degree 1 and 2 recursively, the pathwidth could at most increase by an additive factor of $O(\log n)$. So we assume that G has no vertices of degree 1 or 2 (otherwise we contract the corresponding edges).

First, we prove the lemma for the special case where the maximum degree of G is at most 4, by induction on the number n_4 of vertices of degree 4 in G. If $n_4 = 0$, then $\Delta(G) \leq 3$, and we apply Proposition 10.1. Let us assume that for $n_4 \geq 1$ and for every $\varepsilon > 0$ there exists n_ε such that for every graph with at least n_ε vertices and at most $n_4 - 1$ vertices of degree 4, the lemma holds. Let $v \in V$ be a vertex of degree 4. Every neighbor of v has degree at least 3 (since we contract all the edges incident on the vertices of degree 1 or 2 recursively). Let $i \in \{0, ..., 4\}$ be the number of degree 3 neighbors of v. Thus v has $4 - i$ neighbors of degree 4 and

$$\begin{aligned}
\mathbf{pw}(G) &\leq \mathbf{pw}(G - v) + 1 \\
&\leq \frac{n_3 - i + (4 - i)}{6} + \frac{n_4 - 1 - (4 - i)}{3} + \varepsilon(n - 1) + 1 \\
&\leq \frac{n_3}{6} + \frac{n_4}{3} + \varepsilon n \ .
\end{aligned}$$

Now, suppose that the maximum degree of G is at most 5. We have already proved the base case where $n_5 = 0$. Let us assume that for some $n_5 \geq 1$ the statement of the lemma holds for all graphs with at most $n_5 - 1$ vertices of degree 5, no vertices of degree at least 6 and at least one vertex of degree at most 4. (The case when the graph is 5-regular requires special consideration.)

Let v be a vertex of degree 5. Let us first assume that the graph $G - v$ is not 5-regular. It is clear that $\mathbf{pw}(G) \leq \mathbf{pw}(G - v) + 1$. For $j \in \{3, \ldots, 5\}$ we denote by m_j the number of degree j neighbors of v. By the induction assumption,

$$\mathbf{pw}(G) \leq \mathbf{pw}(G-v)+1 \leq \frac{n_3 - m_3 + m_4}{6} + \frac{n_4 - m_4 + m_5}{3} + \frac{13}{30}(n_5 - 1 - m_5) + 1 + \varepsilon n.$$

For all possible values of $m = (m_3, m_4, m_5)$, we have that

$$\frac{13}{30} \leq \frac{1 + \frac{1}{6}(m_4 - m_3) + \frac{1}{3}(m_5 - m_4)}{1 + m_5}.$$

(The equality is obtained when $m = (m_3, m_4, m_5) = (0, 1, 4)$ which corresponds to the case when v has four neighbors of degree 5 and one of degree 4.) Thus,

$$\mathbf{pw}(G) \leq \frac{n_3}{6} + \frac{n_4}{3} + \frac{13}{30}n_5 + \varepsilon n.$$

If the graph obtained from $G - v$ by contracting edges incident on vertices of degree 1 and 2 is 5-regular, then all neighbors of v in G are of degree 3. Let u be a vertex of degree 5 in $G - v$. Since $G - u - v$ is not 5-regular and $\mathbf{pw}(G) \leq$

$\mathbf{pw}(G-u-v)+2$, we have that

$$\begin{aligned}\mathbf{pw}(G) &\le \mathbf{pw}(G-u-v)+2\\ &\le 2+\frac{n_3-5}{6}+\frac{n_4+5}{3}+\frac{13}{30}(n_5-7)+\varepsilon n\\ &< \frac{n_3}{6}+\frac{n_4}{3}+\frac{13}{30}n_5+\varepsilon n.\end{aligned}$$

Thus the lemma holds for all non 5-regular graphs. Since the removal of one vertex changes the pathwidth by at most one, for sufficiently large n this additive factor of one is dominated by εn, and we conclude that the lemma holds for 5-regular graphs as well.

Using exactly similar arguments one can proceed with the vertices of degree 6. The critical case here is when the vertex of degree 6 has 5 neighbors of degree 6 and one neighbor of degree 5.

For vertices of degree at least 7 we just use the fact that adding a vertex to a graph can increase its pathwidth by at most one.

The inductive arguments used here in combination with Lemma 10.1 can be transformed to a polynomial time algorithm to compute the desired path decomposition. □

The following result bounds treewidth in terms of both the number of vertices and the number of edges and is very useful when we have information about the average degree rather than the maximum degree of the graph.

Lemma 10.2 *For any $\varepsilon > 0$, there exists an integer n_ε such that for every connected graph G with $n > n_\varepsilon$ vertices and $m = \beta n$ edges, $\beta \in [1.5, 2]$, the treewidth of G is at most $(m-n)/3+\varepsilon n$. Moreover, a tree decomposition of the corresponding width can be constructed in polynomial time.*

Proof: First we show the result assuming that the maximum degree $\Delta(G)$ of the graph is bounded by 4 and then extend this result without any degree constraint.

Let n_3 be the number of vertices of degree 3 in G and n_4 be the number of vertices of degree 4 in G. Since the contraction of an edge adjacent to a vertex of degree one and two does not change the treewidth of a graph, we assume that $n_3 = n - n_4$. Thus $\frac{3}{2}n_3 + 2n_4 = \beta n$. Since $n_4 = (2\beta - 3)n$ and $n_3 = (4-2\beta)n$, by

Lemma 10.1 we have that

$$\begin{aligned}\mathbf{tw}(G) &\leq \mathbf{pw}(G) \leq \frac{1}{3}(2\beta-3)n + \frac{1}{6}(4-2\beta)n + \varepsilon n \\ &= \frac{\beta-1}{3}n + \varepsilon n = \frac{m-n}{3} + \varepsilon n.\end{aligned}$$

Now we extend the result without any assumptions on the degrees of the vertices of G. We show this by induction on $n_{\geq 5}$, the number of vertices of degree at least 5. We have already shown that the lemma holds if $n_{\geq 5} = 0$. Let us assume that for $n_{\geq 5} \geq 1$, for every $\varepsilon > 0$ there exist n_ε such that for every graph with at least n_ε vertices and at most $n_{\geq 5} - 1$ vertices of degree at least 5 the lemma holds. Let $v \in V$ be a vertex of degree at least 5. Observe that $G - v$ has at most $m - 5$ edges and that $m - 5 \leq 2(n-1)$. Now we have

$$\begin{aligned}\mathbf{tw}(G) \leq \mathbf{pw}(G) \leq \mathbf{pw}(G-v) + 1 &\leq \frac{m-5-(n-1)}{3} + 1 + \varepsilon n \\ &\leq \frac{m-n}{3} + \varepsilon n = \frac{(\beta-1)n}{3} + \varepsilon n\end{aligned}$$

The inductive arguments used here in combination with Lemma 10.1 can be transformed to a polynomial time algorithm to compute the desired path decomposition.

□

10.3 Branching and Global Application of Width Parameters

In this section we give exact algorithms for MMM, its variants and for #3-Coloring. Our algorithms either branch on a vertex or compute a path decomposition of the original graph. Once it computes a path decomposition, it stops branching and finds the solution of the problem by applying an algorithm based on dynamic programming on graphs of bounded pathwidth/treewidth for the problem.

10.3.1 Minimum Maximal Matching

We first recall the following proposition which is a combination of two classical results due to Moon and Moser [185] and Johnson, Yannakakis and Papadimitriou [154].

Proposition 10.2 ([154, 185]) *Every graph on n vertices contains at most $3^{n/3} = \mathcal{O}(1.4423^n)$ maximal (with respect to inclusion) independent sets. Moreover, all these maximal independent sets can be enumerated with polynomial delay.*

Vertex Cover problem is complement of Independent Set problem, that is, $S \subseteq V$ is a vertex cover of G if and only if $V \setminus S$ is an independent set of G and hence Proposition 10.2 can be used for enumerating minimal vertex covers as well. Our algorithm also uses the following *variation* of characterization of a MMM obtained in the last chapter.

Proposition 10.3 *Let $G = (V, E)$ be a graph and M be a minimum maximal matching of G. Let*

$$V[M] = \{v \mid v \ \in V \textit{and } v \textit{ is an end point of some edge of } M\}$$

be a subset of all endpoints of M. Let $S \subseteq V[M]$ be a vertex cover of G. Let M' be a maximum matching in $G[S]$ and M'' be a maximum matching in $G - V[M']$, where $V[M']$ is the set of the endpoints of edges of M'. Then $X = M' \cup M''$ is a minimum maximal matching of G.

Note that in Proposition 10.3, S does not need to be a *minimal* vertex cover.

Finally, we give a lemma for finding a minimum maximal matching on graphs of bounded treewidth, which we use as a subroutine in our algorithm. The proof of the lemma is based on standard dynamic programming on graphs of bounded treewidth.

Lemma 10.3 *There exists an algorithm to compute a minimum maximal matching of a graph G in time $\mathcal{O}(3^p n^{O(1)})$ when a path decomposition of G of width at most p is given.*

Proof: Let $G = (V, E)$ be a graph with a path decomposition $(X_1, X_2, \ldots, X_l)$ of width p. It is well known [31] that a path decomposition can be turned into a path decomposition $(X_1, X_2, \ldots, X_k)$ where $|X_1| = 1$ and for every $i \in \{1, 2, \ldots, k-1\}$ there is $v \in V$ such that either $X_{i+1} = X_i \cup \{v\}$, or $X_i = X_{i+1} \cup \{v\}$ and such a construction can be done in linear time.

For every node $i \in \{1, 2, \ldots, k\}$, let $S_i \subseteq V$ be the subset of vertices $\bigcup_{j \leq i} X_j$ and $G_i = G[S_i]$.

We use three colors 0, $\hat{1}$ and 1 to represent the state of the vertices in the graph during our dynamic programming algorithm. For each coloring of X_i let $\hat{\mathbf{1}}$ be the set of vertices colored by $\hat{1}$. We compute the minimum size of a matching M in $G_i - \hat{\mathbf{1}}$ where the minimum is taken over all maximal matchings in G_i subject to

- (0): The vertices colored by 0 can not be endpoints of M;
- (1): Every vertex colored 1 is an endpoint of M.

The vertices colored by $\hat{1}$ are not endpoints of M, however we distinguish them from 0 because these vertices will be used as the endpoints of a matching during the next steps.

With every node i of the path decomposition we associate a table which stores the mapping $f_i : \{0, 1, \hat{1}\}^{|X_i|} \to \mathbb{N} \cup \{+\infty\}$. The p columns of the table store possible colorings of the vertices in X_i and for every coloring $c = (c(x_1), ..., c(x_{|X_i|})) \in \{0, 1, \hat{1}\}^{|X_i|}$, we also keep $f_i(c)$ which is the minimum size of a maximal matching for G_i consistent with this coloring.

The dynamic programming starts from X_1 and continues by calculating the table of node $i+1$ from the table of node i. The size of a minimum maximal matching for G is then the minimum value of the function f_k, i.e. the minimum number value of f_k of the table associated with node k.

What follows is a description—for a node i with associated bag X_i—of how the values of the associated table are calculated, depending on the type of i.

$\underline{X_1}$ Suppose that $X_1 = \{x\}$. There are only two possibilities to color x: 0 and $\hat{1}$. The value of f_i is 0 for both colorings.

$\underline{X_{i+1} = X_i \cup \{x\}}$: A coloring $c = (c(x_1), ..., c(x_{|X_i|}), c(x))$ is a valid coloring for X_{i+1} if and only if there exists a valid coloring $c' = (c'(x_1), ..., c'(x_{|X_i|}))$ for X_i such that one of the following conditions holds:

1. $c = c' \times \{0\}$ and for every neighbor z of x in X_{i+1}, either $c(z) = 1$ or $c(z) = \hat{1}$;
2. $c = c' \times \{\hat{1}\}$;
3. There is $z \in X_i : (\forall v \in X_i \setminus \{z\} : c(v) = c'(v))$ and $c(x) = c(z) = 1$ and $c'(z) = \hat{1}$.

Thus

$$f_{i+1}(c) \leftarrow \begin{cases} f_i(c') & \text{if condition 1 holds,} \\ f_i(c') + 1 & \text{if condition 2 or 3 holds,} \\ +\infty & \text{otherwise (i.e. } c \text{ is not valid).} \end{cases}$$

$\underline{X_{i+1} = X_i - \{x\}}$: For every coloring c for X_{i+1}, let $C'(c)$ be the set of colorings for X_i such that for each coloring $c' \in C'(c)$, $c'(x) \neq \hat{1}$ and $\forall z \in X_i$: $c(z) = c'(z)$. The colorings where $c'(x) = \hat{1}$ are not considered, because a decision must be taken for x before it is "forgotten". Consequently, set $f_{i+1}(c) \leftarrow \min_{c' \in C'(c)}\{f_i(c')\}$.

Finally, each table can be computed in time $O(3^p)$ and the overall time complexity of the algorithm is thus $O(3^p n)$. □

Now we are ready to describe our algorithm for MMM which uses Proposition 10.2, Proposition 10.3 and Lemma 10.3 as subroutines. Our detailed algorithm is depicted in Figure 10.1. The algorithm of Figure 10.1 outputs a minimum maximal matching of G. The parameter G of the Algorithm `findMMM` always corresponds to the original input graph, $H = (V_H, E_H)$ is a subgraph of G and C is a vertex cover of $G - V_H$. To solve MMM we run `findMMM(`$G, G, \emptyset$`)`. The algorithm consists of three parts.

Branch (lines 1–5). The algorithm branches on a vertex v of maximum degree and returns the matching of minimum size found in the two subproblems created according to the following rules:

(R1) Vertices $N(v)$ are added to the vertex cover C and $N[v]$ is deleted from H;

(R2) Vertex v is added to the vertex cover C and v is deleted from H.

```
Algorithm findMMM($G, H, C$)
Input: A graph $G$, an induced subgraph $H$ of $G$ and a set of vertices
       $C \subseteq V(G) - V(H)$.
Output: A minimum maximal matching of $G$ subject to $H$ and $C$.
if ($\Delta(H) \geq 4$) or ($\Delta(H) = 3$ and $|C| > 0.17385|V(G)|$) then
    choose a vertex $v \in V(H)$ of maximum degree
    $M_1 \leftarrow$ findMMM($G, H - N[v], C \cup N(v)$)                    (R1)
    $M_2 \leftarrow$ findMMM($G, H - \{v\}, C \cup \{v\}$)                  (R2)
    return the set of minimum size among $\{M_1, M_2\}$
else if ($\Delta(H) = 3$ and $|C| \leq 0.17385|V(G)|$) or ($\Delta(H) \leq 2$ and
$|C| \leq 0.31154|V(G)|$) then
    output a path decomposition of $G$ using Lemma 10.4
    Then find a minimum maximal matching, M, using Lemma 10.3 and
    return $M$.
    The Algorithm stops.
else
    $X \leftarrow E(G)$
    foreach minimal vertex cover $Q$ of $H$ do
        $M' \leftarrow$ a maximum matching of $G[C \cup Q]$
        Let $V[M']$ be the set of end points of $M'$
        $M'' \leftarrow$ a maximum matching of $G[C \cup V(H) \setminus V[M']]$
        if $M' \cup M''$ is a maximal matching of $G$ and $|X| > |M' \cup M''|$ then
            $X \leftarrow M' \cup M''$
    return $X$
```

Figure 10.1: Algorithm findMMM(G, H, C)

Compute path decomposition (lines 6–8). The algorithm outputs a path decomposition using Lemma 10.4 (discussed later). Then the algorithm finds a minimal maximal matching using the pathwidth algorithm of Lemma 10.3 on graphs of bounded pathwidth.

Enumerate minimal vertex covers (lines 9–17). The algorithm enumerates all minimal vertex covers of H. For every minimal vertex cover Q of H, $S = C \cup Q$ is a vertex cover of G and the characterization of Proposition 10.3 is used to find a minimum maximal matching of G.

The conditions under which these different parts of the algorithm are executed have been carefully chosen to optimize the overall running time of the algorithm, including the pathwidth algorithm of Lemma 10.3. Note that a path decomposition is computed at most once in an execution of the algorithm as the algorithm stops right after outputting the path decomposition and then apply Lemma 10.3 on this path decomposition. Also note that the minimal vertex covers of H are only enumerated in a leaf of the search tree corresponding to the recursive calls of the algorithm.

We define a *branch node* of the search tree of the algorithm to be a recursive call of the algorithm. A branch node is uniquely identified by the triple (G, H, C), that form the parameters of `findMMM`. Now we give a theorem which proves the correctness and the time complexity of the algorithm.

Theorem 10.1 *A minimum maximal matching of a graph on n vertices can be found in time $\mathcal{O}(1.4082^n)$.*

Proof: We first show the correctness of the algorithm and then bound its running time.

Correctness: The algorithm either branches on a vertex of degree at least 3, or produces a path decomposition of G, or enumerates minimal vertex covers as subroutines. The correctness of the step where a path decomposition of G is computed follows from Lemma 10.3 and the correctness of the branching step is obvious.

If minimal vertex covers of H are enumerated, then the algorithm finds a set of edges X by making use of Proposition 10.3 for all possible sets $S = C \cup Q$, where

Q is a minimal vertex cover of H. Consider a subset of vertices $Q' \subseteq V_H$ such that $C \cup Q'$ forms the set of end points of a minimum maximal matching M of G. Observe that Q' is a vertex cover of H. Hence if $Q \subseteq Q'$ is a minimal vertex cover of H then $S = C \cup Q$ is a vertex cover of G and by applying Proposition 10.3 for S, a minimum sized maximal matching is obtained for G. Hence, all the minimal vertex covers H are enumerated as possible candidates for Q.

Time Analysis: For the running time analysis, it is essential to prove a good bound on the width of the path decomposition of G, obtained by the algorithm. We start with a lemma.

Lemma 10.4 *Let $G = (V, E)$ be the input graph and (G, H, C) be a branch node of the search tree of our algorithm. The pathwidth of the graph is bounded by $\mathbf{pw}(H) + |C|$. In particular,*
(a) If $\Delta(H) \leq 3$, then $\mathbf{pw}(G) \leq (\frac{1}{6} + \varepsilon)|V_H| + |C|$ for any $\varepsilon > 0$.
(b) If $\Delta(H) \leq 2$, then $\mathbf{pw}(G) \leq |C| + 2$.
A path decomposition of the corresponding width can be found in polynomial time.

Proof: Let $T = V_G \setminus (V_H \cup C)$ be the set of vertices the algorithm removed from H which were not included in C. Note that the set of endpoints of a maximal matching forms a vertex cover of G. Thus when Algorithm `findMMM` decides that a vertex v is not in the vertex cover C, i.e. when it places it in T, all its neighbors are included in C. Hence, for every branch node (G, H, C) of the search tree of the algorithm we have the following (a) *T is an independent set*, and (b) $N[V_H] \cap T = \emptyset$. Hence the pathwidth of $G[V_H \cup T]$ is equal to $\mathbf{pw}(H)$. Given a path decomposition P of $G[V_H \cup T]$, we obtain a path decomposition P' of G by adding C to every bag of P. The pathwidth of P' is therefore bounded by $\mathbf{pw}(H) + |C|$. The remaining part of the lemma follows from Proposition 10.1 and the fact that graphs with maximum degree at most 2 have pathwidth at most 2. □

Let us resume the proof of the theorem. Let $\alpha = 0.17385$ and $\beta = 0.31154$. First, consider the conditions under which a path decomposition may be computed. By combining the pathwidth bounds of Lemma 10.4 and the running time of the algorithm of Lemma 10.3, we obtain that MMM can be solved in time

$$\mathcal{O}^*\left(\max\left(3^{(1+5\alpha)/6}, 3^{\beta}\right)^n\right)$$

when the path decomposition part of the algorithm is executed.

Assume now that the path decomposition part is not executed. Then, the algorithm continues to branch when the maximum degree $\Delta(H)$ of the graph H is 3. And so, $|C| > \alpha n$ when $\Delta(H)$ first becomes 3. At this point, the set C has been obtained by branching on vertices of degree at least 4 and we investigate the number of subproblems obtained so far. Let L be the set of nodes in the search tree of the algorithm that corresponds to subproblems where $\Delta(H)$ first becomes 3. Note that we can express $|L|$ by a two parameter function $A(n,k)$ where $n = |V_G|$ and $k = \alpha n$. This function can be upper bounded by a two parameter recurrence relation corresponding to the unique branching rule of the algorithm:

$$A(n,k) = A(n-1,k-1) + A(n-5,k-4)$$

When the algorithm branches on a vertex v of degree at least 4 the function is called on two subproblems. If v is not added to C (**(R1)**), then $|N[v]| \geq 5$ vertices are removed from H and $|C|$ increases by $|N(v)| \geq 4$. If v is added to C (**(R2)**), then both parameters decrease by 1.

Let r be the number of times the algorithm branches in the case **(R1)**. Observe that $r \in [0, k/4]$. Let L_r be a subset of L such that the algorithm has branched exactly r times according to **(R1)** in the unique paths from the root to the nodes in L_r. Thus, $|L|$ is bounded by $\sum_{i=0}^{k/4} |L_i|$.

To bound the number of nodes in each L_i, let $l \in L_r$ and P_l be the unique path from l to the root in the search tree. Observe that on this path the algorithm has branched $k-4i$ times according to **(R2)** and i times according to **(R1)**. Hence, the length of the path P_l is $k-3i$. By counting the number of sequences of length $k-3i$ where the algorithm has branched exactly i times according to **(R1)**, we get $|L_i| \leq \binom{k-3i}{i}$. Thus if the path decomposition is not computed, the time complexity $T(n)$ of the algorithm is

$$\begin{aligned} T(n) &= \mathcal{O}^*\left(\sum_{i=0}^{k/4}\binom{k-3i}{i}T'(n-5i-(k-4i))\right) \\ &= \mathcal{O}^*\left(\sum_{i=0}^{k/4}\binom{k-3i}{i}T'(n-i-k)\right) \qquad (10.1) \end{aligned}$$

where $T'(n')$ is the time complexity to solve a problem on a branch node (G, H, C) in L with $n' = |V_H|$. (Let us remind that in this case the algorithm branches on vertices of degrees 3 and enumerates minimal vertex covers of H.) Let $p = (\beta - \alpha)n$. To bound $T'(n')$, we use similar arguments as before and use Proposition 10.2 to bound the running time of the enumerative step of the algorithm. Thus we obtain the following.

$$\begin{aligned} T'(n') &= \mathcal{O}^*\left(\sum_{i=0}^{p/3}\binom{p-2i}{i}3^{\frac{n'-4i-(p-3i)}{3}}\right) \\ &= \mathcal{O}^*\left(3^{(n'-p)/3}\sum_{i=0}^{p/3}\binom{p-2i}{i}3^{-i/3}\right) \qquad (10.2) \end{aligned}$$

We bound $T(n')$ by $\mathcal{O}(3^{(n'-p)/3}d^p)$ for some constant $d \in (1, 2)$, the value of d will be determined later. Substituting this in (10.1), we get:

$$\begin{aligned} T(n) &= \mathcal{O}^*\left(\sum_{i=0}^{k/4}\binom{k-3i}{i}3^{\frac{n-i-k-p}{3}}d^p\right) \\ &= \mathcal{O}^*\left(3^{(1-\beta)n/3}d^p\sum_{i=0}^{k/4}\binom{k-3i}{i}3^{-i/3}\right). \end{aligned}$$

Further suppose that $\sum_{i=0}^{k/4}\binom{k-3i}{i}3^{-i/3}$ sums to $\mathcal{O}(c^k)$ for a constant $c \in (1, 2)$, then the overall time complexity of the algorithm is bounded by

$$\mathcal{O}^*\left(\left(3^{(1-\beta)/3}d^{\beta-\alpha}c^{\alpha}\right)^n\right) .$$

Claim 10.1 *$c < 1.3091$ and $d < 1.3697$.*

Proof: The sum over binomial coefficients $\sum_{i=0}^{k/4} \binom{k-3i}{i} 3^{-i/3}$ is bounded by $(k/4)B$ where B is the maximum term in this sum. Let as assume that $B = \binom{k-3i}{i} 3^{-i/3}$ for some $i \in \{1, 2, \dots, k/4\}$. To compute the constant c, let $r := c - 1$. We obtain

$$B = \binom{k-3i}{i} 3^{-i/3} \quad \leq \quad \frac{(1+r)^{k-3i}}{r^i} 3^{-i/3}.$$

Here we use the well known fact that for any $x > 0$ and $k \in \{0, \cdots, n\}$,

$$\binom{n}{k} \leq \frac{(1+x)^n}{x^k} \quad .$$

By choosing r to be the minimum positive root of $\frac{(1+r)^{-3}}{r} 3^{-1/3} - 1$, we arrive at $B < (1+r)^k$ for $r \in (0.3090, 0.3091)$. Thus $c < 1.3091$. The value of d is computed in a similar way as we computed c. □

Finally, we get the following running time for Algorithm `findMMM` by substituting the values for α, β, c and d:

$$\mathcal{O}^* \left(\max \left(3^{(1-\beta)/3} d^{\beta-\alpha} c^{\alpha}, 3^{(1+5\alpha)/6}, 3^{\beta} \right)^n \right) = \quad \mathcal{O}(1.4082^n) \ .$$

□

10.3.2 Some variations of MMM

In this subsection we give exact algorithms for two problems which are closely related to Minimum Maximal Matching.

Our algorithm for MEDS depends on an old result which shows that every minimum maximal matching is a MEDS [128].

Proposition 10.4 ([128]) *Let $G = (V, E)$ be a graph. Then every minimum maximal matching of G is a minimum edge dominating set.*

Proposition 10.4 in connection with Theorem 10.1 gives us the following corollary.

Corollary 10.1 *A* Minimum Edge dominating Set *of a graph with n vertices can be found in time $\mathcal{O}(1.4082^n)$.*

Matrix Domination reduces to finding a MEDS in a bipartite graph [128]. We obtain an improved algorithm for Matrix Domination by using the fact that all minimal vertex covers of a triangle free graph (and a bipartite graph in particular) on n vertices can be listed in time $\mathcal{O}^*(2^{n/2})$ [41]. The proof of the following theorem is similar to the one of Theorem 10.1.

Theorem 10.2 *Given a matrix M of size $m \times n$ with entries in $\{0,1\}$,* Matrix Domination *can be solved in time $\mathcal{O}(1.3918^{m+n})$.*

Proof: To solve Matrix Domination we solve MMM on a bipartite graph on $n+m$ vertices. As observed by Byskov [41], all minimal vertex covers of a triangle free graph (and a bipartite graph in particular) on $n+m$ vertices can be listed in time $\mathcal{O}^*(2^{(n+m)/2})$. Thus the minimal vertex covers of the graph H in Algorithm `findMMM` can be listed faster, and similar to Theorem 10.1 we can estimate the running time of the algorithm in this case by balancing the running time of the algorithm based on a path decomposition of the graph with

$$\mathcal{O}^*\left(\left(2^{(1-\beta)/2}d^{\beta-\alpha}c^{\alpha}\right)^{n+m}\right) \tag{10.3}$$

where $\mathcal{O}(d^{(\beta-\alpha)(n+m)})$ is solution to $\sum_{i=0}^{p/3}\binom{p-2i}{i}2^{-i/2}$ while $\mathcal{O}(c^{\alpha(n+m)})$ is solution to $\sum_{i=0}^{k/4}\binom{k-3i}{i}2^{-i/2}$. The values we get for the constants are: $\alpha \leq 0.16110$, $\beta \leq 0.30091$, $d \leq 1.3744$, $c \leq 1.3127$. Substituting these values in (10.3), we obtain the claim of the theorem. □

10.3.3 Counting 3-Colorings (#3-Coloring)

Recall that a *proper coloring* of a graph is an assignment of colors to its vertices such that no edge is monochromatic. Given a graph $G = (V, E)$, Coloring problem asks for a proper coloring of V, minimizing the number of colors used on the vertices. The problem of 2-Coloring, that is, can the given graph be colored with at most 2 colors is polynomial time solvable (bipartite graph testing) but r-Coloring is NP-complete for any $r \geq 3$ [127]. Here we look at the counting version of 3-Coloring which is defined below.

> 3-Coloring: Given a graph $G = (V, E)$ find a function $c : V \rightarrow \{R, B, G\}$ such that for every $\{u, v\} \in E$ we have $c(u) \neq c(v)$.

We denote by #3-Coloring the problem to count all 3-colorings of a graph. Our algorithm for #3-Coloring is very similar to the one presented for MMM. Here we give a simple description for the algorithm.

We associate two colors $\{R, BG\}$, with every vertex initially. If we decide to color a vertex v with R then we color its neighbors BG and remove $N[v]$ from the graph otherwise we color v with BG and remove it from the graph. Let C_1 be the set of vertices of G which are colored R and let $C_2 = V \setminus C_1$ be the vertices colored BG. Now G has a 3-coloring respecting this precoloring if and only if $G[C_2]$ is bipartite and the number of 3-colorings respecting this precoloring is the number of 2-colorings of $G[C_2]$ which is 2^t, where t is the number of connected components of $G[C_2]$. Hence, given a fixed precoloring of a graph G with R and BG, we can compute the number of 3-colorings respecting this precoloring in polynomial time.

We also need the following well known dynamic programming algorithm on graphs with bounded treewidth for our algorithm.

Lemma 10.5 *[28] Given a graph $G = (V, E)$ with a tree decomposition of G of width ℓ, #3-Coloring can be solved in time $\mathcal{O}(3^{\ell} n^{O(1)})$.*

As in the algorithm for MMM we have three phases in the algorithm. Here H is the induced subgraph on uncolored vertices of G at some recursive step in the algorithm.

Branch. The algorithm branches on a vertex v of maximum degree in H and returns the sum of #3-Coloring found in the two subproblems created according to the following rules:

- **(R1)** the vertices in $N(v)$ are added to the color class C_2, v is added to the color class C_1 and $N[v]$ is deleted from H;
- **(R2)** the vertex v is added to the color class C_2 and v is deleted from H.

Compute path decomposition. If the maximum degree of H is at most 3 and the size of C_2 is at most $0.3342n$ or if the maximum degree of H is at most 2 and the size of C_2 is at most $0.44517n$, then this is a step for applying pathwidth algorithm on the original graph. At this point, the algorithm outputs a path decomposition and the algorithm stops without backtracking. Then #3-Coloring is solved using the pathwidth algorithm of Lemma 10.5 on the original graph G.

Enumerate 2-colorings of H. When the maximum degree of H is at most 2 and the size of C_2 does not satisfy the conditions of path decomposition phase then the algorithm enumerates all possible two colorings with R and BG of H to get the coloring of whole graph G with R and BG.

Let us observe here that the analysis and the algorithm for #3-Coloring remains the same as that of the MMM algorithm (Figure 10.1) except the role of C in the algorithm for MMM is taken by C_2 in the algorithm for #3-Coloring. If we replace C with C_2 in Lemma 10.4 then we get the same upper bounds on the pathwidth of the original graph G. In the algorithm for #3-Coloring we enumerate all proper 2 colorings of H. This is different from enumerating maximal independent sets of H as we did in the algorithm for MMM. This change leads to the use of different α and β than in MMM to optimize the running time of the algorithm for #3-Coloring. Let $T(n)$ denote the time taken by the algorithm for #3-Coloring on graphs on n vertices. For a fixed $\alpha \le 0.3342$, $\beta \le 0.44517$, we fix $k = \alpha n$ and $p = (\beta - \alpha)n$. Then the running time of the algorithm is bounded by $T(n)$ below when pathwidth algorithm is not used in the graph.

$$\begin{aligned}
T(n) &= \mathcal{O}^*\left(\sum_{i=0}^{k/4} \binom{k-3i}{i} T'(n-5i-(k-4i))\right) \\
&= \mathcal{O}^*\left(\sum_{i=0}^{k/4} \binom{k-3i}{i} T'(n-i-k)\right),
\end{aligned}$$

and

$$\begin{aligned}
T'(n') &= \mathcal{O}^*\left(\sum_{i=0}^{p/3} \binom{p-2i}{i} 2^{n'-4i-(p-3i)}\right) \\
&= \mathcal{O}^*\left(2^{(n'-p)} \sum_{i=0}^{p/3} \binom{p-2i}{i} 2^{-i}\right).
\end{aligned}$$

Here $T'(n')$ is the running time of the algorithm on H when its maximum degree is 3 and the size of C_2 (vertices colored with BG) is at least $0.3342n$. We bound $T(n')$ by $\mathcal{O}^*(2^{(n'-p)}d^p)$ for some constant $d \in (1, 2)$, the value of d will be determined later. Substituting this in the expression for $T(n)$, we get

$$\begin{aligned} T(n) &= \mathcal{O}^*\left(\sum_{i=0}^{k/4}\binom{k-3i}{i}2^{n-i-k-p}d^p\right) \\ &= \mathcal{O}^*\left(2^{(1-\beta)n}d^p\sum_{i=0}^{k/4}\binom{k-3i}{i}2^{-i}\right). \end{aligned}$$

Further suppose that $\sum_{i=0}^{k/4}\binom{k-3i}{i}2^{-i}$ sums to $\mathcal{O}(c^k)$ for a constant $c \in (1, 2)$, then the overall time complexity of the algorithm is bounded by

$$\mathcal{O}^*\left(\left(2^{(1-\beta)}d^{\beta-\alpha}c^{\alpha}\right)^n\right) .$$

Similar to the analysis in the algorithm for MMM, we get values for α, β, c and d. Finally, substituting the values for $\alpha = 0.3342$, $\beta = 0.44517$, $c = 1.2538$ and $d = 1.2972$, we get the running time for the algorithm for #3-Coloring as

$$\mathcal{O}^*\left(\max\left(2^{(1-\beta)}d^{\beta-\alpha}c^{\alpha}, 3^{(1+5\alpha)/6}, 3^{\beta}\right)^n\right) = \mathcal{O}(1.6308^n) .$$

This gives us the following theorem.

Theorem 10.3 *Let $G = (V, E)$ be an undirected graph on n vertices, then #3-Coloring can be solved in $\mathcal{O}(1.6308^n)$ time.*

This improves the $\mathcal{O}(1.770^n)$ time algorithm presented in [123].

10.4 Branching & Local Application of Width Parameters

10.4.1 Weighted Vertex Cover

Here we apply our technique to design a simple fixed parameter tractable algorithm for the parameterized version of Weighted Vertex Cover problem.

- k-Weighted Vertex Cover (k-WVC): Given a graph $G = (V, E)$, a weight function $w : V \to \mathbb{R}^+$ such that for every vertex v, $w(v) \geq 1$ and $k \in \mathbb{R}^+$, find if exists a vertex cover of weight at most k, where the weight of a vertex cover C is $w(C) = \sum_{v \in C} w(v)$.

Now, we present an algorithm that combines Branch & Reduce and dynamic programming on graphs of bounded treewidth. It is well known that a minimum vertex cover can be found in time $\mathcal{O}(2^\ell n^{O(1)})$ in a graph of treewidth at most ℓ.

Proposition 10.5 ([28]) *Given a graph G with weights on its vertices and a tree decomposition of G of width at most ℓ, a minimum weighted vertex cover of G can be found in time $\mathcal{O}(2^\ell n^{O(1)})$.*

We need *kernelization* for our algorithm for weighted vertex cover. We state the proposition on kernelization of [192] that we use in our algorithm.

Proposition 10.6 ([192]) *Let $G = (V, E)$ be a graph, $w : V \to \mathbb{R}^+$ such that for every vertex v, $w(v) \geq 1$ and $k \in \mathbb{R}^+$. There is an algorithm that in time $\mathcal{O}(kn + k^3)$ either concludes that G has no vertex cover of weight at most k, or outputs a kernel of size at most $2k$ (graph with at most $2k$ vertices).*

First we apply Proposition 10.6 to obtain a kernel of size at most $2k$. Then, as long as the maximum degree of the graph is at least 4, the algorithm branches on a vertex v of maximum degree; two subproblems are created according to the following rules:

(1) add v to the partially constructed vertex cover and delete v from the graph;

(2) add $N(v)$ to the partially constructed vertex cover and delete $N[v]$ from the graph.

If the maximum degree of the graph is at most 3, then by Proposition 10.1, a tree decomposition of small treewidth $\frac{2k}{6} = \frac{k}{3}$ (tw) can be found on the kernel of size $2k$ in polynomial time and we can use a $\mathcal{O}(2^{tw}n^{O(1)})$ dynamic programming algorithm to solve k-Weighted Vertex Cover. The correctness is clear from the presentation and the running time of the algorithm is dominated by the following recurrences on $T(k)$.

$$\begin{array}{rcll} T(k) & \leq & T(k-1) + T(k-4), & \text{[Branching Step]} \\ T(k) & \leq & 2^{2k/6}n^{O(1)} & \text{[Treewidth Step].} \end{array}$$

Though the gap between the solutions of the above two recurrences is huge, it is hard to balance them. The problem is that the known bound on the size of the kernel remains fixed even though the average degree or the maximum degree of the graph decreases. Our algorithm takes exponential space as size of tables used in the dynamic programming algorithm for k-WVC is exponential in k. This results in the following theorem.

Theorem 10.4 k-WVC *on a graph on n vertices can be solved in time* $\mathcal{O}(1.3803^k n^{O(1)})$ *and space* $\mathcal{O}(1.2599^k n^{O(1)})$.

This simple algorithm is comparable to the best known parameterized algorithm for weighted vertex cover which runs in time $\mathcal{O}(1.3788^k n^{O(1)})$ and space $\mathcal{O}(1.3603^k n^{O(1)})$ [192] that involves a lot of cases.

10.5 Parameterized Edge Dominating Set and its Variants

In this section we show that the technique branching & global application of width parameters can be used to obtain the fastest known parameterized algorithm for the following problem.

- k-Weighted Edge Dominating Set (k-WEDS): Given a graph $G = (V, E)$, a weight function $w : E \to \mathbb{R}^+$ such that for every edge e, $w(e) \geq 1$ and $k \in \mathbb{R}^+$, find a set of edges $D \subseteq E$ of weight $w(D) = \sum_{e \in D} w(e)$ at most k such that every edge of $E \setminus D$ is adjacent to an edge in D.

Observe that if a graph G has an edge dominating set of weight at most k then it has a vertex cover of weight at most $p = 2k$. As in the algorithm for MMM, we construct a partial vertex cover C of G by branching on vertices of maximum degree. As observed by Fernau [112], if $G \setminus C$ has maximum degree one, then an optimum edge dominating set compatible with C for G can be found in polynomial time. Using this and branching on vertices of degree at least 2, Fernau obtains an algorithm with running time $\mathcal{O}(2.6181^k n^{O(1)})$.

Here we branch on vertices of degree at least 3. That is we pick a vertex v of degree at least 3 and include either v or $N(v)$ in C. This gives us the following recurrence on p: $T(p) = T(p-1) + T(p-3)$, which solves to $\mathcal{O}(1.465572^p) = \mathcal{O}(2.1480^k)$.

Now, suppose that the algorithm has reached a branch node (G, H, C) of the recurrence tree and $\Delta(H) \leq 2$. Then by Lemma 10.4, the pathwidth of the graph G is bounded by $|C| + 1$. Again, we use a different strategy based on the size of $|C|$ at the branch node. If $|C| \leq \alpha p$ (α to be determined later) then we compute a path decomposition of width ℓ and apply an algorithm with running time $\mathcal{O}(3^\ell n^{O(1)})$ similar to the algorithm of Theorem 10.3 to obtain a minimum edge dominating set. Otherwise, the algorithm continues branching on vertices of degree 2 of H in time $1.6181^{p-\alpha p}$. To obtain the optimal value of α, we solve the equation $1.465572^{\alpha p} 1.6181^{p-\alpha p} = 3^{\alpha p}$ and obtain $\alpha = 0.4018$. This gives us a running time of $\mathcal{O}(1.55501^p n^{O(1)}) = \mathcal{O}(2.4181^k n^{O(1)})$ for k-Edge Dominating Set.

We observed in Section 10.3.2 that an exact algorithm for Edge Dominating Set also implies exact algorithms for Minimum Maximal Matching and Matrix Domination. Thus, we obtain the following result for k-Weighted Edge Dominating Set and its related problems.

Theorem 10.5 k-Weighted Edge Dominating Set*,* k-Minimum Maximal Matching *and* k-Matrix Domination *can be solved in time* $\mathcal{O}(2.4181^k n^{O(1)})$.

10.6 Conclusion

Branching and dynamic programming on graphs of bounded treewidth are very powerful techniques to design efficient exact algorithms. In this chapter, we combined these two techniques in different ways and obtained improved exact algorithms for #3-Coloring, MMM and its variants. We also applied the technique to design fixed parameter tractable algorithms and obtained fast algorithms for k-WVC and k-WEDS which also shows the versatility of our technique. The most important aspects of this technique are that the resulting algorithms are very elegant and simple while at the same time the analysis of these algorithms is non-trivial.

11 Maximum r-Regular Induced Subgraph Problems

The problem of finding a Maximum Induced Subgraph having properties like acyclicity [115, 201], bipartiteness [41, 199], regularity [46, 47, 116, 208, 215] and regularity with dominance [35] is among the fundamental problems in graph algorithms. Here we study one such problem, namely the Maximum r-Regular Induced Subgraph problem, where r is a fixed positive integer. The problem is defined as follows:

> Maximum r-Regular Induced Subgraph (M-r-RIS): Given an undirected graph $G = (V, E)$, find a maximum subset of vertices $R \subseteq V$ such that the induced subgraph on R, $G[R]$, is r-regular.

When r is 0 or 1, the problem corresponds to the well studied Maximum Independent Set and Maximum Induced Matching problems respectively. While Maximum Independent Set problem is among the six classical NP-complete problems [127], Maximum Induced Matching problem was introduced by Stockmeyer and Vazirani in [216] who showed it to be NP-complete [216]. But only recently, has it been shown [47] that the problem of finding a maximum sized r-regular induced subgraph is NP-complete for any value of r.

M-r-RIS problems find various applications in other combinatorial problems. An exact algorithm to find a maximum sized independent set is used as a subroutine in algorithms to find a minimum sized Feedback Vertex Set [115], Odd Cycle Transversal [199] etc. Algorithms for enumerating maximal independent sets in a graph have been at the heart of exact algorithms for several problem as seen in Chapter 11 and also in [41, 170, 199]. Similarly, finding large induced matching is a subroutine in finding *strong edge-coloring* of a graph [46, 91, 96, 215]. M-r-RIS problems with some property $\mathcal{P}$ like dominance find applications in game theory [35].

In this chapter we look at the M-r-RIS problems (a) from exact exponential time algorithm paradigm and (b) from the view point of combinatorial bounds on the number of maximal r-regular induced subgraphs possible on a graph on n vertices.

An exact algorithm to find a Maximum Independent Set or M-0-RIS problem has attracted a lot of attention in the area of exact exponential time algorithms [116, 208] and the current fastest known exact algorithm runs in time $\mathcal{O}(1.2108^n)$ [208]. However, there is no algorithm better than $\Theta(2^n)$ is known for larger values of r.

Here, we give a simple generic algorithm for Maximum r-Regular Induced Subgraph problems taking $\mathcal{O}(c^n)$ time, $c < 2$, a constant, depending on r alone. As a corollary, we obtain $\mathcal{O}(1.6957^n)$, $\mathcal{O}(1.7069^n)$ and $\mathcal{O}(1.7362^n)$ time algorithms for Maximum Induced Matching, Maximum 2-Regular Induced Subgraph and Maximum Induced Cubic Subgraph problems respectively. We then generalize the algorithm to solve the counting and enumerating version of M-r-RIS problems in the same time.

An interesting consequence of our algorithm is that it gives an algorithmic upper bound of $o(2^n)$ on the number of maximal r-regular induced subgraphs on graphs on n vertices, if r is some constant. We then investigate the lower bounds on the number of maximal r-regular induced subgraphs a graph may have and observe that for large values of r, the lower bounds and the upper bounds (mentioned above) on the number of maximal r-regular induced subgraphs on n vertices are "almost identical". For small values of r, we improve the upper bounds using a different technique and give a matching lower and upper bounds on the number of maximal r-regular induced subgraphs. The bounds generalize the result of Moon and Moser [185] who showed an upper and lower bound of $3^{n/3}$ on the number of maximal independent sets on a graph on n vertices.

We also give applications of the algorithms developed in this chapter to design non trivial exact algorithms for a special case of Induced Subgraph Isomor-

PHISM problem, that is INDUCED r-REGULAR SUBGRAPH ISOMORPHISM problem, where r is a constant, MAXIMUM BOUNDED DEGREE INDUCED SUBGRAPH problems, δ-SEPARATING MAXIMUM MATCHING problem and EFFICIENT EDGE DOMINATING SET problem.

All our algorithms are simple but their analyzes are non trivial. These algorithms are based on one of the most important and widely used tool of exact algorithms, namely the *Branch & Reduce* paradigm. We also use a new measure, other than the number of vertices or edges to measure the progress of the algorithms and use it extensively in many of our upper bound proofs. Measures other than the number of vertices have been a source of many recently developed non trivial exact algorithms [115, 116, 201].

Organization of the rest of the chapter

In Section 11.1, we give a *generic* algorithm for MAXIMUM r-REGULAR INDUCED SUBGRAPH problems and then generalize it to solve the counting and enumerating version of the problems.

In Section 11.2 we give matching lower and upper bounds on the number of maximal r-regular induced subgraphs for various values of r. r exact algorithms for M-r-RIS problems for $r = 1$ and 2. We also obtain non trivial exact algorithms for EFFICIENT EDGE DOMINATING SET and INDUCED r-REGULAR SUBGRAPH ISOMORPHISM problems in this Section.

In Section 11.3 we develop faster exact algorithms for M-r-RIS problems for $r = 1$ and 2 and also obatain a non trivial exact algorithm for δ-SEPARATING MAXIMUM MATCHING problem.

We conclude with some remarks and open problems in Section 11.4

In the rest of the chapter, we assume that all our graphs are simple and undirected. Given a graph $G = (V, E)$, n represents the number of vertices, and m represents the number of edges. For a subset $V' \subseteq V$, by $G[V']$ we mean the subgraph of G induced on V'. By $N(u)$ we represent all vertices (excluding u) that are adjacent to u, and by $N[u]$, we refer to $N(u) \cup \{u\}$. Similarly, for a subset $D \subseteq V$, we define $N[D] = \cup_{v \in D} N[v]$. $N_i(v)$ is the set of vertices such that for every vertex $u \in N_i(v)$, the shortest distance from u to v is i, ie, $\{u \mid d(u, v) = i\}$. Similarly $N_i[v] = \{u \mid d(u, v) \leq i\}$. By $N_G[v]$ $(N_G(v))$ we mean $N[v]$ $(N(u))$ in the graph G.

In our algorithm unless we state otherwise $N(v)$ and $N[v]$ mean $N_G(v)$ and $N_G[v]$ respectively.

11.1 Maximum r-Regular Induced Subgraph

Our algorithm is based on the Branch & Reduce paradigm. It selects a vertex v and on one branch of recursion grows a maximum r-regular induced subgraph without v and on the other a maximum r-regular induced subgraph containing v and then outputs the one with the maximum size. At any point of time in our algorithm we maintain a set R (of possible vertices of a M-r-RIS) and construct one connected component of this R. Once we finish one connected component, say R_i, we remove all the neighbors of vertices of R_i which are not in R_i, that is $N[R_i] - R_i$, from the graph and then proceed. Based on the structure of $G[R]$, we divide our algorithm into two phases:

1. Active Phase : $G[R]$ is $\emptyset$ or an r regular induced subgraph.
2. Growth Phase : There exists a unique component R_i of $G[R]$ such that $G[R_i]$ is not an r regular subgraph.

In Active Phase we initiate constructing a new connected component for the possible M-r-RIS. We select a vertex v and at one branch construct a solution not including v and at other branches we construct a solution containing v and a r-subset of $N(v)$. This leads to $\binom{|N(v)|}{r} + 1$ way branching. In the Growth Phase, we choose a vertex v of the unique component R_i of $G[R]$ ($G[R_i]$ is not a r regular subgraph) such that degree of v in $G[R_i]$ is $r_v < r$ and branch on all possible subsets of size $r - r_v$ of $N(v) - R$, which leads to $\binom{|N(v)-R|}{r-r_v}$ way branching.

At any point of time, our algorithm has a 4 tuple $(G' = (V', E'), G, r, R)$. Here, G' contains the unexplored vertices (vertices which are neither in R nor those which have been removed from consideration). G is the initial input graph. This graph never changes during recursion and is only used for checking whether or not $G[R]$ is induced r-regular. R is a set of vertices already chosen for a possible maximum r-regular induced subgraph. We return $-\infty$ if we detect that the corresponding branch can not lead to an r-regular induced subgraph; for example if in the Growth Phase, if we find a vertex $v \in R$ has degree r_v in $G[R]$ but strictly less than $r - r_v$ neighbors in V'. The details of our algorithm are presented in Figure 11.1.

Theorem 11.1 *Let $G = (V, E)$ be a graph on n vertices and $r \geq 1$ be a fixed constant. Then there exists a constant c, $c < 2$ such that the* Maximum r-Regular Induced Subgraph *problem can be solved in $\mathcal{O}(c^n)$ time.*

Proof: The correctness of the algorithm is clear. The analysis of time complexity is involved and we present the details here.

From now onwards let r be a fixed positive constant. Observe that the above algorithm is guided by the following recurrences:

$$\begin{aligned} T(n) &\leq T(n-1) + \max_d \left\{ \binom{d}{r} T(n-d-1) \right\}, \text{ for any } d \geq r, \\ &\qquad\qquad\qquad\qquad\qquad\qquad \text{[Active Phase].} \qquad (11.1) \\ T(n) &\leq \binom{d}{t} T(n-d) \qquad\qquad d \geq t, \quad 1 \leq t \leq r-1 \\ &\qquad\qquad\qquad\qquad\qquad\qquad \text{[Growth Phase].} \qquad (11.2) \end{aligned}$$

The smallest positive root x of the following inequalities,

$$h_d(x, r) = x^{d+1} - x^d - \binom{d}{r} \geq 0 \ , d \geq r$$

and

$$g_d(x, t) = x^d - \binom{d}{t} \geq 0 \ \ , d \geq t, \quad 1 \leq t \leq r-1,$$

are solutions to the above recurrences. It is clear that $x = 2$ satisfies these inequalities. Now we show that if r is a constant then we can find a c, a function of r alone, and $c < 2$ satisfying these set of inequalities. We need the following easy lemma for our proof.

Lemma 11.1 *For any $r \geq 5$, $\binom{2r}{r} \leq \frac{2^{2r}}{4}$.*

```
Algorithm Max-$r$-RIS ($G' = (V', E'), G, r, R$)

Step 1: [active phase] If $G[R]$ is not $r$ regular and not empty then
        go to Step 2.

        Step 1a: Obtain a new $G'$ by removing $N[R]$ from $G'$.

        Step 1b: Remove all vertices of degree $< r$ recursively from
            $G'$.

        Step 1c: If $G'$ is non empty then select a vertex $v$ of maximum
            degree $d \geq r$ and branch in following ways: (1) $v \notin R$, and
            (2) $v \in R$ and then a set $S \subseteq N_{G'}(v)$ such that $|S| = r$ in
            $R$.

            1. $R_1 \leftarrow$ Max-r-RIS($G' - v, G, r, R$)
            2. for ($S \subseteq N_{G'}(v)$ & $|S| = r$),
               $R_S \leftarrow$ Max-r-RIS($G' - N_{G'}[v], G, r, R \cup S \cup \{v\}$).

            return the set (or the number) of maximum size between
                $\{R_1\}$ and $\{R_S \mid S' \subseteq N_{G'}(v)\ |S'| = r\}$.

Step 2: [growth phase] Let $R'$ be the unique component of $G[R]$
        such that $G[R']$ is not a $r$ regular induced subgraph. $R_1 \leftarrow$
        $-\infty$. Choose a vertex $v$ with degree say $r_i$ in $G[R']$ such that
        $1 \leq r_i \leq r - 1$ and $|N(v) \cap V'| \geq r - r_i$.

        1. for ($S \subseteq (N(v) \cap V')$ & $|S| = r - r_i$ & maximum degree
           of $G[R' \cup S]$ is $\leq r$ )
           $R_S \leftarrow$Max-r-RIS($G' - (N(v) \cap V'), G, r, R \cup S$)

        return the set (or the number) of maximum size between
            $\{R_1\}$ and $\{R_{S'} \mid S' \subseteq (N(v) \cap V')$ & $|S'| = r - r_i\}$.
```

Figure 11.1: A Generic Algorithm to find a Maximum r-Regular Induced Subgraph

Proof:

$$\begin{aligned}
\binom{2r}{r} &= \frac{1\cdot 2\cdot 3\cdots 2r}{(1\cdot 2\cdot 3\cdots r)(1\cdot 2\cdot 3\cdots r)}\\
&= \frac{(2\cdot 4\cdots 2r)(1\cdot 3\cdots(2r-1))}{r!r!}\,\frac{(2\cdot 4\cdots 2r)}{(2\cdot 4\cdots 2r)}\\
&= \frac{2^r r!\;2^r r!}{r!r!}\,\frac{(1\cdot 3\cdots(2r-1))}{(2\cdot 4\cdots 2r)}\\
&\le \frac{2^{2r}}{4} \qquad\qquad \text{(for } r\ge 5\text{)}.
\end{aligned}$$

□

Observe that

$$x^d-\binom{d}{r}\ge x^d(x-1)-\binom{d}{r}\ge x^{d+1}-x^d-\binom{d}{r}.$$

The inequality holds as $x\le 2$. This shows that if there exists $c=f(r)$ such that $h_d(f(r),r)\ge 0$ then $g_d(f(r),r)\ge 0$. Hence we concentrate on the polynomials coming from the ACTIVE PHASE as they represent the dominating recurrences.

Now we show that if there exists a $c=f(r)$ such that $h_{2r}(c,r)\ge 0$ then we can choose a c' such that $h_d(c',r)\ge 0$ for any d. We take $c'=\max\left\{c,\frac{2r+1}{r+1}\right\}$. We prove this using forward induction for $d\ge 2r$ and backward induction for $d\le 2r$. For the base case of forward induction, observe that $h_{2r}(c',r)\ge h_{2r}(c,r)\ge 0$. Now assume that $h_d(c,r)\ge 0$ for some $d\ge 2r$. Then

$$\begin{aligned}
h_{d+1}(c',r) &= c'^{d+2}-c'^{d+1}-\binom{d+1}{r}\\
&= c'(c'^{d+1}-c'^{d})-\binom{d+1}{r}\\
&\ge c'\binom{d}{r}-\binom{d+1}{r} \quad \text{(from induction hypothesis)}\\
&\ge \frac{2r+1}{r+1}\binom{d}{r}-\binom{d+1}{r}\\
&\ge 0.
\end{aligned}$$

The last inequality follows because

$$\frac{2r+1}{r+1} \geq \frac{d+1}{d+1-r} = \frac{\binom{d+1}{r}}{\binom{d}{r}} \text{ for } d \geq 2r.$$

Similarly using backward induction we will show that $h_d(c', r) \geq 0$ for $d \leq 2r$. For the base case of backward induction, observe that $h_{2r}(c', r) \geq h_{2r}(c, r) \geq 0$. Now assume that $h_d(c, r) \geq 0$ for some $d \leq 2r$. Then

$$\begin{aligned} h_{d-1}(c', r) &= c'^{d} - c'^{d-1} - \binom{d-1}{r} \\ &= \frac{1}{c'}(c'^{d+1} - c'^{d}) - \binom{d-1}{r} \\ &\geq \frac{1}{c'}\binom{d}{r} - \binom{d-1}{r} \quad \text{(from induction hypothesis)} \\ &\geq \frac{1}{2}\binom{d}{r} - \binom{d-1}{r} \quad \text{(since } c' \leq 2) \\ &\geq 0. \end{aligned}$$

The last inequality follows because

$$\frac{1}{2} \geq \frac{d-r}{d} = \frac{\binom{d-1}{r}}{\binom{d}{r}} \text{ for } d \leq 2r.$$

Observe that for $r \geq 0$, $1 \leq \frac{2r+1}{r+1} < 2$, is a constant depending on r alone. So now we are left with showing a $c = f(r)$ for $h_{2r}(x, r)$. For $r \geq 5$, we know that $\binom{2r}{r} \leq \frac{2^{2r}}{4}$. We choose a c such that $c^{2r+1} - c^{2r} \geq \frac{2^{2r}}{4}$ which will prove the desired result. We take $c = 2^{1-\frac{1}{2r}}$ for $r \geq 5$ and $c = 1.761$ for $r \leq 4$. For small values of r we get the desired number by directly solving the corresponding equations.

Hence for any $r \geq 0$, we choose

$$c = \max\left\{1.761,\ 2^{1-\frac{1}{2r}},\ \frac{2r+1}{r+1}\right\}.$$

This proves that our generic algorithm **Max-r-RIS** takes $\mathcal{O}(c^n)$ time, $c < 2$, for any positive constant r. □

We gave a conservative bound on the value of c in the Theorem 11.1, as our main aim there was to obtain a $c < 2$ for any fixed constant r. For smaller values of r, we obtain improved bounds on c by directly finding the roots of the polynomials coming from the recurrences 11.1 and 11.2 of MAX-r-RIS algorithm. Without going into the details, we list c for various values of r in the Table 11.1, where $\mathcal{O}(c^n)$ is the runtime of our **Max-r-RIS** algorithm.

$r =$	1	2	3	4	5	6
$c =$	1.69562	1.70688	1.73615	1.76357	1.78554	1.80351
$r =$	7	8	9	10	15	20
$c =$	1.81846	1.83111	1.84195	1.85136	1.88452	1.90486
$r =$	30	50	75	100	125	150
$c =$	1.92868	1.95138	1.96458	1.97186	1.97652	1.97979

Table 11.1: Improved Upper Bounds on c for Various r

We observe that the **Max-r-RIS** algorithm can be generalized to solve the counting versions of M-r-RIS problems. The counting version of M-r-RIS problems (#M-r-RIS) asks for the number of maximum r regular induced subgraphs of the given graph G. We can also consider counting the number of maximal r-regular induced subgraphs of the given graph G which we call #MAXIMAL-r-RIS problems. To solve these problems we allow our algorithm **Max-r-RIS** to enumerate all the R's it finds during the recursion for G and check whether they are maximal if we want to count maximal r-regular induced subgraphs alone. If we want to count maximum r-regular induced subgraphs then we also need to check the size of R. Thus we get the following theorem.

Theorem 11.2 *Let $G = (V, E)$ be a graph on n vertices and $r \geq 1$ be a fixed constant. Then (a) #M-r-RIS problems and (b) #MAXIMAL-r-RIS problems can be solved in $\mathcal{O}(c^n)$ time, where c is max of* $\{1.761, 2^{1-\frac{1}{2r}}, (2r+1)/(r+1)\}$.

We observed above that our algorithm enumerates all maximal r-regular induced subgraphs. Hence Theorem 11.2 also implies that the number of maximal r-regular induced subgraphs of a graph on n vertices is upper bounded by the time complexity of the algorithm. Let $\mathscr{M}_r(n)$ denote the number of maximal r-regular induced subgraph in a graph on n vertices, then we get following theorem.

Theorem 11.3 *Let $G = (V, E)$ be a graph on n vertices and $r \geq 1$ be a fixed constant. Then $\mathscr{M}_r(n)$ is upper bounded by c^n, where c is max of*

$$\{1.761, 2^{1-\frac{1}{2r}}, (2r+1)/(r+1)\},$$

i.e. $\mathscr{M}_r(n)$ is upper bounded by $o(2^n)$, if r is a fixed constant.

In the next section we consider the lower bounds on the number of maximal r-regular induced subgraphs on graphs on n vertices and improve the upper bounds coming from Theorem 11.3 to match the lower bounds for various r. But before we move on to the next section we give an application of Theorem 11.3.

11.1.1 Induced r-Regular Subgraph Isomorphism

Here we consider a special case of Induced Subgraph Isomorphism (Ind-SI) problem.

> Induced Subgraph Isomorphism: Given a graph $G = (V, E)$, $|V| = n$ and H, the question is to determine whether there exists a $H' \subseteq V$ such that $G[H'] \cong H$.

A brute force algorithm for this is to enumerate all subsets H' of size $|H|$ of G and check whether $G[H'] \cong H$, using the $O(n^{o(n)})$ time graph isomorphism algorithm [16] resulting in overall $O(2^n n^{o(n)})$ runtime. The question is: *can we do this in $\mathcal{O}(c^n)$ time where $c < 2$ is a constant?* Here, we answer this question for a special class of H, that is when H is a r-regular graph with r being a constant. Even with such restrictions this problem is NP-hard as it contains problems like Independent Set. We show the following theorem.

Theorem 11.4 *Given a graph $G = (V, E)$ on n vertices and a graph H, where H is r-regular, for a constant r, we can determine whether there exists a $H' \subseteq V$ such that $G[H'] \cong H$ in $\mathcal{O}(c^n)$ time, where $c < 2$ is a constant depending only on r.*

Proof: Let $H = \{H_1, H_2, \cdots, H_q\}$ where each H_i is a connected component of H. If there exists a $H' \subseteq V$ such that $G[H'] \cong H$ then H' can also be written as

$\{H'_1, H'_2, \cdots H'_q\}$ where H'_i is a connected component of $G[H']$, such that $G[H'_i] \cong H_i$ for $1 \leq i \leq q$.

The crucial observation is that if there exists a H' such that $G[H'] \cong H$ then there exists a maximal r-regular induced subgraph R extending H' such that each of the connected components of H' appears as a connected component of $G[R]$. By applying Theorem 11.3, we enumerate all maximal r-regular induced subgraphs of a graph on n vertices in $\mathcal{O}(c^n)$ time, $c < 2$ a constant depending on r alone. Now given a R, a maximal r-regular induced subgraph of G, we check the isomorphism of each connected component of $G[R]$ with each of H_i using the polynomial time bounded degree graph isomorphism algorithm of Luks [175]. If we obtain a H' such that $G[H'] \cong H$ then we return H' else we return no. The correctness of the algorithm follows now. The runtime of the algorithm is dominated by the number of maximal r-regular induced subgraphs enumerated by Theorem 11.3 which is $\mathcal{O}(c^n)$ where $c < 2$ is a constant depending only on r. □

11.2 Bounds on Number of Maximal r-Regular Induced Subgraphs

Moon and Moser [185] gave a matching lower and upper bound of $3^{n/3}$ on the number of maximal independent sets on a graph on n vertices. We generalize this result and give matching algorithmic lower and upper bounds on $\mathscr{M}_r(n)$ for larger values of r.

11.2.1 Bounds on $\mathscr{M}_1(n)$ or Number of Maximal Induced Matching

For lower bound assume that $n \equiv (0 \bmod 5)$. Consider the graph $G = \bigcup_{i=1}^{\frac{n}{5}} K_5^i$ that is $n/5$ disjoint copies of K_5 (K_n^i represents the complete graph on n vertices). Observe that we need to include one edge from each copy of the K_5 (we can include exactly one edge from each copy) to obtain a maximal induced matching for G. Since a K_5 has 10 edges and for any K_5 we can select any edge, we get $10^{n/5}$ distinct maximal induced matching for G, giving a lower bound of $10^{n/5}$ on $\mathscr{M}_1(n)$. This shows the following theorem.

Theorem 11.5 *There exists a graph G such that* $\mathscr{M}_1(n)$ *is at least* $10^{n/5} \approx 1.58489^n$.

For an upper bound proof, we obtain recurrences for $\mathscr{M}_1(n)$ by considering various cases based on the maximum degree of the graph, improving on the bound of 1.69562 of Theorem 11.1.

Theorem 11.6 $\mathscr{M}_1(n)$ *is at most* $10^{n/5} \approx 1.58489^n$ *and all the maximal induced matching of a graph* G *can be enumerated in time proportional to* $10^{n/5}$.

Proof: We give recurrences based on the maximum degree of the graph. Our recurrences are based on the considerations *that a maximal induced matching either contains a vertex* v *or not* and we give the upper bound by combining the number of maximal induced matchings possible in both the cases. We count maximal induced matchings containing v by counting maximal induced matching which contains edges of the form (v, u), where $u \in N(v)$, for all the neighbors u of v.

Case A (v is a vertex of maximum degree d in G and $d \geq 5$): The number of maximal induced matching of G containing an edge (v, u), $u \in N(v)$ is upper bounded by $\mathscr{M}_1(n - (d + 1))$, as no other neighbors of v can be part of a maximum induced matching which contains the edge (v, u). Since v has d neighbors we get:

$$\mathscr{M}_1(n) \leq \mathscr{M}_1(n-1) + d\mathscr{M}_1(n-(d+1)) \qquad d \geq 5.$$

Case B (v be a vertex of maximum degree of G and its degree is 4): Here we have two cases, (a) $N_2(v) = \emptyset$ or (b) $N_2(v) \neq \emptyset$. In the first case, v and its four neighbors form a connected component by themselves, say $\mathcal{C}_v$, with v having degree 4. So the number of maximal induced matching in G is maximized if $\mathcal{C}_v$ is a connected graph on 5 vertices having maximum number of induced matching. This happens when $\mathcal{C}_v$ is K_5. This gives us following recurrence:

$$\mathscr{M}_1(n) \leq 10\mathscr{M}_1(n-5).$$

In the case (b), we have a vertex $u \in N(v)$ such that it has a neighbor x such that $x \notin N[v]$. Now if we consider the case where we are counting the number of maximal induced matching containing (v, u) we get that it is upper bounded by $\mathscr{M}_1(n-6)$ as neighbors of v, u can not be part of these maximal induced matchings. So we get

$$\mathscr{M}_1(n) \leq \mathscr{M}_1(n-1) + 3\mathscr{M}_1(n-5) + \mathscr{M}_1(n-6).$$

Case C (v be a vertex of maximum degree of G and its degree is 3): This case requires a detailed case analysis. Let us assume that the neighbors of v are w_1, w_2, w_3. Here we have following cases which we apply in the order of their appearance.

Case 1 ($N_2(v) = \emptyset$): As arguments in Case B, we can take $\mathcal{C}_v = K_4$, leading to the following recurrence:

$$\mathscr{M}_1(n) \leq 6\mathscr{M}_1(n-4).$$

Case 2 ($N_2(v) \neq \emptyset$ **&** v **has at least** 1 **neighbor, say** w_1 **such that the degree of** w_1 **is** 1**:**) Consider the branch where we count maximal induced matching without containing v, that is number of maximal matchings in the graph $G - v$, degree of w_1 becomes 0 and hence number of maximal matching in $G - v$ is same as in $G - v - w_1$. Also either w_2 or w_3, say w_2, has a neighbor $x \notin N[v]$. Hence when we count maximal induced matching containing (v, w_2) it amounts to counting maximal induced matching in $G - N[v] - x$. These observations give us the following recurrence:

$$\mathscr{M}_1(n) \leq \mathscr{M}_1(n-2) + 2\mathscr{M}_1(n-4) + \mathscr{M}_1(n-5).$$

Case 3 ($N_2(v) \neq \emptyset$ **&** v **has at least** 2 **neighbors, say** w_1, w_2 **which have neighbors not in** $N[v]$ **:**) In this case we get the following recurrence by branching on the vertex v.

$$\mathscr{M}_1(n) \leq \mathscr{M}_1(n-1) + \mathscr{M}_1(n-4) + 2\mathscr{M}_1(n-5).$$

Case 4: ($N_2(v) \neq \emptyset$ **&** v **has exactly** 1 **neighbor, say** w_1**, which has**

neighbors not in $N[v]$**.)** Here we have the following cases based on the structure of $G[N(v)]$.

Case 4a: (w_1 **has at least two neighbor** $x_1, x_2 \notin N[v]$**.)** Observe that now we have an edge (w_2, w_3) (as no neighbor of v has degree 1 because of Case 2) and there is no edge between w_2 and w_1 or w_3 and w_1 as w_1 has already degree 3 with v, x_1 and x_2 as its three neighbors. When we look for a maximal induced matching in $G-v$ then (w_2, w_3) becomes an isolated edge and every maximal induced matching of $G - v$ contains (w_2, w_3) which gives us the following recurrence:

$$\mathscr{M}_1(n) \leq \mathscr{M}_1(n-3) + 2\mathscr{M}_1(n-4) + \mathscr{M}_1(n-6).$$

Case 4b: (w_1 **has exactly one neighbor** $x_1 \notin N[v]$**.)** Now we definitely have an edge (w_2, w_3), as both of them can't be neighbors of w_1 (otherwise w_1 will have degree 4) and no neighbor of v has degree 1 because of Case 2. Suppose there is no edge from w_2 or w_3 to w_1 then as in Case 4a we get following recurrence:

$$\mathscr{M}_1(n) \leq \mathscr{M}_1(n-3) + 2\mathscr{M}_1(n-4) + \mathscr{M}_1(n-5).$$

So now we assume that there is an edge from either w_2 or w_3 to w_1, say we have (w_2, w_1) as an edge. Now we consider the case where we are looking for a maximal induced matching in $G-v$. Observe that in this case (w_3, w_2), or (w_2, w_1) or (w_1, x_1) must be part of every maximal induced matching of $G - v$. Suppose not and let M be a maximal induced matching of $G-v$ which does not contain any of these edges. Since w_1 is not an end point of any edge in M (as the only edges incident on w_1 are (w_1, x_1) and (w_1, w_2)), we can extend M by adding the edge (w_2, w_3) and maintaining that M remains an induced matching. This contradicts the maximality of M. Hence in the case where we want to count the maximal induced matching without containing v we count the ones containing at least one of these edges as one of them must be part of every maximal induced matching of $G - v$. When $(w_2, w_3) \in M$, w_1 is removed from the graph G and since v is already been removed from the graph it amounts to counting maximal induced matching in $G - N[v]$ which is upper bounded by $\mathscr{M}_1(n-4)$. Similarly when $(w_1, w_2) \in M$ or $(w_1, x) \in M$, we obtain that the number of maximal induced matchings is bounded by $\mathscr{M}_1(n-5)$ in both the cases. So in this case the recurrence for $\mathscr{M}_1(n)$ is given by:

$$\mathscr{M}_1(n) \leq 3\mathscr{M}_1(n-4) + 3\mathscr{M}_1(n-5).$$

This completes the description for degree 3 case.

Case D (Maximum degree of G is at most 2): We can assume that there is no vertex of degree 0 as they do not contribute to a maximal induced matching. Similarly we can also assume that there are no isolated edges as they are part of every maximal induced matching. Now if there is a connected component on exactly three vertices then either it is a path of length 2 or a triangle. Every maximal induced matching will contain one and exactly one edge from these components on three vertices, leading to the following worst case recurrence in this case:

$$\mathscr{M}_1(n) \leq 3\mathscr{M}_1(n-3).$$

So now we assume that every connected component has at least 4 vertices which could be a path $\mathcal{P}$ of length at least 3 or a cycle $\mathcal{C}$ of length at least 4. If we have path $\mathcal{P}$ then branching on the second vertex v of the path (neighbor of a degree 1 vertex) gives us:

$$\mathscr{M}_1(n) \leq \mathscr{M}_1(n-2) + \mathscr{M}_1(n-3) + \mathscr{M}_1(n-4).$$

Similarly if we have cycle $\mathcal{C}$ then branching on any vertex of $\mathcal{C}$ gives us the following recurrence:

$$\mathscr{M}_1(n) \leq \mathscr{M}_1(n-1) + 2\mathscr{M}_1(n-4).$$

The dominating recurrence among all the above recurrences, obtained by solving them using mathematical package like MATLAB, is

$$\mathscr{M}_1(n) \leq 10\mathscr{M}_1(n-5)$$

which solves to $10^{n/5}$ giving us the required upper bound on the $\mathscr{M}_1(n)$.

The above proof can be made algorithmic by making each of the different recurrences as branching steps. This observation gives that we can enumerate all maximal induced matching in polynomial delay in $\mathcal{O}(10^{n/5})$ time. □

Now we give a simple application of Theorem 11.6.

11.2.2 Efficient Edge Dominating Set

Efficient Edge Dominating Set (EEDS) problem is defined as follows:

> EEDS : Given a graph $G = (V, E)$, find a minimum sized subset $D \subseteq E$ (if exists) such that every edge of G has a vertex in common with exactly one edge of D.

Grinstead et al. [137] showed this problem NP-hard. Moreover, Georges et al. [129] showed that any efficient edge dominating set is a maximum induced matching, though not every MIM is an efficient edge dominating set. We use Theorem 11.6 to enumerate all maximal induced matchings and check whether that forms an efficient edge dominating set. This way not only we solve the decision version of the problem but also find an EEDS if it exists. As an application of Theorem 11.6, we get the following.

Theorem 11.7 *Given a graph $G = (V, E)$ on n vertices, we can determine whether there exists an* Efficient Edge Dominating Set *and find one if it exists in $\mathcal{O}(10^{n/5})$ time.*

11.2.3 Bounds on $\mathscr{M}_r(n)$ for $r \geq 2$

Now we obtain matching upper and lower bounds for larger values of $r(\geq 2)$. To give the upper bound on $\mathscr{M}_r(n)$, we define the following generalized problem.

GEN-r-RIS (G-r-RIS): Given a graph $G = (V, E)$ and $R \subseteq V$, such that $G[R]$ is a connected induced subgraph of degree at most r. The objective is to find an $R' \subseteq V - R$ such that $G[R \cup R']$ is a r regular subgraph extending R and R' is of maximum size among such subsets.

Observe that given any instance (G, R), where R satisfies the constraints in the definition of G-r-RIS problem, if we can give a bound on the number of R' such that $G[R' \cup R]$ is a maximal r-regular subgraph then by setting $R = \emptyset$ we have an upper bound on $\mathscr{M}_r(n)$. Given an instance (G, R) where R satisfies the constraints in G-r-RIS problem, we define μ as follows:

$$\mu = \alpha|N^R| + \beta|U|.$$

Here $N^R = N[R] - R$ and $U = V - N[R]$. In other words, we assign a weight of α to the vertices of N^R and β to the vertices of U. The value of α and β depend on the problem. The weight of a vertex changes in following situation:

1. If a vertex goes to N^R from U then the weight changes from β to α and the μ changes by $\delta = \beta - \alpha$.

2. If a vertex has current weight either α or β and the vertex is either included in R or removed from the graph then the weight changes to 0. In this case μ changes either by α or β.

We use μ as a measure to capture the progress of the algorithm and obtain recurrences as a function of μ, rather than n, the number of vertices and also give an upper bound on $\mathscr{M}_r(n)$ as a function f of μ. We exemplify the approach by giving the matching lower and upper bound on the number of *maximal* 2*-regular induced subgraphs.*

Theorem 11.8 *There exists a graph G such that $\mathscr{M}_2(n)$ is at least $35^{n/7} \approx 1.66181^n$. On the other hand for $\mathscr{M}_2(n)$ is at most $35^{n/7} \approx 1.66181^n$. Moreover, all the maximal* 2*-regular induced subgraphs of a graph G can be enumerated in time proportional to $35^{n/7}$.*

Proof: For the lower bound on $\mathscr{M}_2(n)$, assume that $n \equiv (0 \bmod 7)$ and consider the graph $G = \bigcup_{i=1}^{\frac{n}{7}} K_7^i$, $n/7$ disjoint copies of K_7. Any maximal 2-regular induced subgraph of G contains a 2 regular induced subgraph (a triangle) from each copy of K_7. Every K_7 has 35 distinct triangles and hence G has $35^{n/7}$ distinct maximal 2-regular induced subgraphs. This shows the desired lower bound on $\mathscr{M}_2(n)$.

For the upper bound, we consider the generalized problem where we have been given $(G = (V, E), R)$ and $G[R]$ is a connected induced subgraph of degree at most r. We give a bound on the number of R''s, i.e. the size of the set

$$\left\{ R' \;\middle|\; G[R' \cup R] \text{ is a maximal 2-regular } \right\}$$

as a function f of μ. Depending on various cases we give recurrence relation for f.

We assume that the minimum degree of G is at least 2, as the vertices of degree at most 1 can never be part of any maximal 2 regular induced subgraphs and hence can be removed.

Case 1: $(G[R] \neq \emptyset)$ Here we have two cases based on the degree of vertices in $G[R]$. For a subset $X \subseteq V$, by $deg_X(v)$ we mean the number of neighbors of v in $G[X]$. Suppose we have a vertex $v \in R$ such that $deg_R(v) = 2$ and has l neighbors in $V - R$ then

$$f(\mu) \leq f(\mu - \alpha l);$$

as none of the l neighbors of v in $V - R$ can be selected in any R' extending R and hence can be removed from the graph, leading to a decrease in μ by at least αl. Now suppose there is a vertex v such that degree of v is $d \geq 2$ in G and $deg_R(v) = 1$.

Now any maximal 2-regular induced subgraph extending R must contain one of the neighbors of v in $V - R$. Hence when we include a neighbor u of v in R we remove all other neighbors of v from G as they can not be part of any R' extending R. This reduces μ by $\alpha(d - 1)$. Since there are $d - 1$ neighbors of v in $V - R$, we get the following recurrence:

$$f(\mu) \leq (d - 1)f(\mu - \alpha(d - 1)).$$

Case 2: ($R = \emptyset$ **and** $\exists$ **a vertex with degree** ≥ 7) We first note that every vertex has weight β in this case. Let v be a vertex of maximum degree d. A maximal 2-regular induced subgraph of G either does not contain v or contains v and its two neighbors. In the first case μ reduces by β and in the other cases where v and its two neighbors are selected in R and other neighbors of v are removed from the graph, μ decreases by $(d+1)\beta$. This gives the following worst case recurrence on $f(\mu)$:

$$f(\mu) \leq f(\mu - \beta) + \binom{d}{2} f(\mu - (d+1)\beta).$$

When $d \geq 7$ this recurrence itself gives us the desired bound on $\mathscr{M}_2(n)$. So from now on we assume that the maximum degree of G is at most 6. To obtain the desired bound in this case we refine the recurrences on $f(\mu)$ based on following three cases. These cases are applied in order of their appearance.

(a) Con-Com Case: There exists a vertex $v \in G$ such that $G[N[v]]$ is one of the connected component of G. Call the connected component containing v $\mathcal{C}_v$. Now the number of maximal 2-regular induced subgraphs of G is maximized when we have $\mathcal{C}_v$ such that $\mathcal{C}_v$ has the maximum number of maximal 2-regular induced subgraphs. This happens precisely when $\mathcal{C}_v = K_t$ where $t = deg_V(v)$. So for this case we get:

$$f(\mu) \leq \binom{d+1}{3} f(\mu - \beta(d+1)), \text{ for some } d \text{ such that } 2 \leq d \leq 6.$$

(b) Cut-Edge Case: We have a vertex v such that it has a unique neighbor u having a unique neighbor x such that $x \notin N[v]$. Since the edge (u,x) is a cut edge it is not part of any maximal 2-regular induced subgraph. So the number of maximal 2 regular subgraphs of G is upper bounded by the number of maximal 2 regular subgraphs of G' obtained from G by removing the edge (u,x). This reduces it to Con-Com Case.

(c)At-Least-2-In-$N_2[v]$ Case: In this case every vertex $v \in V$ either has a neighbor u such that u has at least 2 neighbors not in $N[v]$ or there are at least two neighbors of v which don't have neighbors in $N[v]$. For this case we give a generic recurrence. Partition the neighbor set $N(v)$ of v into W_1, W_2 and W_3 such that every vertex $u \in W_1$ has $N(u) \subseteq N[v]$, each vertex in W_2 has an unique neighbor x such that $x \notin N[v]$ while every vertex $u \in W_3$ has at least 2

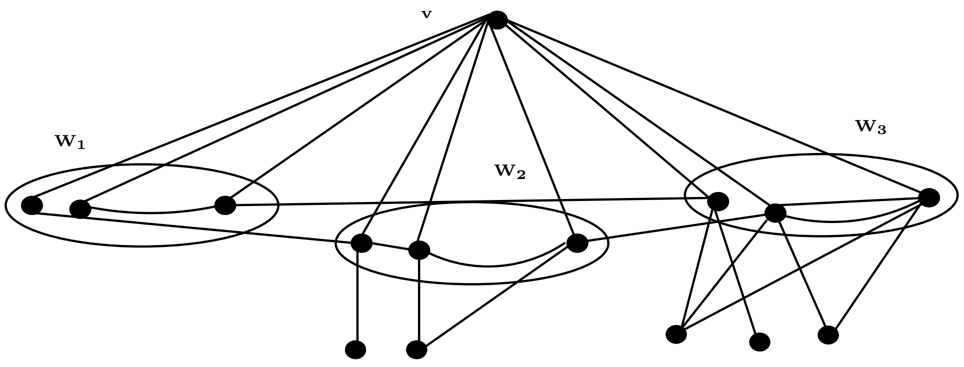

Figure 11.2: An Illustration of Partition Set used in At-Least-2-In-$N_2[v]$ Case.

neighbors not in $N[v]$. See Figure 11.2. By S_y^v we mean the set $N(y)$-$N[v]$. Let $2 \leq \sum_{i=1}^{3} |W_i| = d \leq 6$. We consider the recurrence on $f(\mu)$ based on whether or not v is a part of a maximal 2-regular induced subgraph. When $v \notin R$ μ changes to $\mu - \beta$. Now we consider the case when v and its two neighbors u_1, u_2 and $u_1 \neq u_2$ are in R and see the change in μ based on which W_i's, $1 \leq i \leq 3$, u_1 and u_2 belong.

(A) $[(\mathbf{u_1}, \mathbf{u_2}) \in \mathbf{W_1} \times \mathbf{W_1}]$ μ changes to $\mu - \beta(d+1)$.

(B) $[(\mathbf{u_1}, \mathbf{u_2}) \in \mathbf{W_1} \times \mathbf{W_2}]$ The only way we can have a 2-regular induced subgraph is when (u_1, u_2) is an edge and v, u_1, u_2 is a triangle. This implies that x, the unique neighbor of u_2 not in $N[v]$ will be removed from the graph. This reduces μ to $\mu - \beta(d+1) - \beta$.

(C) $[(\mathbf{u_1}, \mathbf{u_2}) \in \mathbf{W_1} \times \mathbf{W_3}]$ Similar to the previous case we can argue that μ at least reduces to $\mu - \beta(d+1) - 2\beta$.

(D) $[(\mathbf{u_1}, \mathbf{u_2}) \in \mathbf{W_2} \times \mathbf{W_2}]$ The worst case is when u_1 and u_2 have a common neighbor x which is not in $N[v]$. In this case μ changes to $\mu - \beta(d+1) - \beta$.

(E) $[(\mathbf{u_1}, \mathbf{u_2}) \in \mathbf{W_2} \times \mathbf{W_3}]$ If (u_1, u_2) is an edge or u_1 and u_2 have a common neighbor x then either $\{v, u_1, u_2\}$ or $\{v, u_1, u_2, x\}$ forms a 2 regular induced subgraph leading to a reduction of $\beta(d+1) - 2\beta$ in μ. When none of these cases arise then since x is an unique neighbor of u_1, x gets included in R and two neighbor of u_2 become elements of N^R, leading to change in μ by $\beta(d+1) - \beta - 2\delta$.

(F) $[(\mathbf{u_1}, \mathbf{u_2}) \in \mathbf{W_3} \times \mathbf{W_3}]$ Here the worst case is when u_1 and u_2 have exactly two neighbors not in $N[v]$ and $S^v_{u_1} = S^v_{u_2}$, that is u_1 and u_2 have common neighbors not in $N[v]$. This reduces μ by $\beta(d+1) - 2\delta$ as both neighbors of u_1 and u_2 which are not in $N[v]$ become elements of N^R.

Above discussion gives us the following recurrence on $f(\mu)$.

$$\begin{aligned} f(\mu) \;\leq\; & f(\mu-\beta) + \binom{|W_1|}{2} f(\mu-\beta(d+1)) + |W_1||W_2| f(\mu-\beta(d+1)-\beta) \\ & + |W_1||W_3| f(\mu-\beta(d+1)-2\beta) + \binom{|W_2|}{2} f(\mu-\beta(d+1)-\beta) \\ & + |W_2||W_3| f(\mu-\beta(d+1)-\beta-2\delta) + \binom{|W_3|}{2} f(\mu-\beta(d+1)-2\delta). \end{aligned}$$

We assume that $\binom{l_1}{l_2} = 0$ if $l_1 < l_2$. Note that, $|W_1| \leq d-1$ and if there is a unique neighbor u of v having a neighbor x such that $x \notin N[v]$ then $W_2 = \emptyset$ because of the Cut-Edge Case.

We numerically obtain $\alpha = 1.45$, $\beta = 2$ and $\delta = \beta - \alpha = 0.55$, as values which minimize the above set of recurrences on f.

We used a program to generate the above set of recurrences based on different partitions of $N(v)$ and found that the worst case recurrence among the above set after setting $\alpha = 1.45$ and $\beta = 2$ corresponds to the following scenario:

$$d = 5,\ W_1 = W_2 = \emptyset \ \text{ and } \ \forall (y,z) \in W_3 \times W_3,\ |S^v_y \cup S^v_z| = 2.$$

The recurrence corresponding to this scenario is:

$$f(\mu) \leq f(\mu-\beta) + 10 f(\mu - 6\beta - 2\delta).$$

All the recurrences occurring in all the above cases (Cases 1 & 2) are dominated by

$$f(\mu) \leq \binom{7}{3} f(\mu - 7\beta)$$

which solves to $(35)^{\frac{\mu}{7\beta}}$. Now given a graph G, $\mu(G) \leq n\beta$, and hence

$$\mathscr{M}_2(n) \leq f(\beta n) \leq 35^{\beta n/7\beta} = 35^{n/7}.$$

This proves the required upper bound. These cases can be changed in branching steps leading to an enumeration algorithm running in $\mathcal{O}(35^{n/7}) = \mathcal{O}(1.66181^n)$ time. □

To obtain a lower bound on $\mathscr{M}_r(n)$ for larger values of r we need to find a function $g(r)$ such that when we take G as $\frac{n}{g(r)}$ disjoint copies of $K_{g(r)}$, $\binom{g(r)}{r+1}^{1/g(r)}$ is maximized. We obtain the following description for $g(r)$. The proof of Lemma 11.2 is based on estimates on binomial coefficients.

Lemma 11.2 *Given r, $g(r)$ defined below*

$$g(r) = \begin{cases} 2r+3 & 0 \le r \le 11 \\ 2r+4 & 12 \le r \le 100 \\ 2r+2+\left\lfloor \frac{1}{2}\ln\left(\frac{(2r+1)\pi}{2}\right) + O\left(\frac{(\ln r)^2}{r}\right)\right\rfloor & r > 100 \end{cases}$$

maximizes $\binom{g(r)}{r+1}^{1/g(r)}$.

Proof: We give the proof of the lemma in two stages and assume that $r > 100$ as the description for $g(r)$, $r \le 100$, can easily be obtained using hands on calculations or by using symbolic algebra packages.

In the first stage we show that $g(r)$ is upper bounded by $2r+2+O(\ln r)$ and then in the second stage we refine it to obtain the desired description for $g(r)$.

Lemma 11.3 *For larger values of r, $g(r)$ is asymptotically equal to $2r+j$, where $1 \le j \le 10\ln r$.*

Proof: We show this claim using contradiction. Hence we assume that $J > 10\ln r$ and show that if $t = 2r+J$ then $\binom{t}{r}^{1/t} \ge \binom{t+1}{r}^{1/t+1}$ which will show the desired bound on j. Now we first show some simple claims which we make use of.

Claim 11.1 *If k is large then $k^{1/k} < 1 + \frac{3\ln k}{k}$.*

Proof: We know that $e^x \le 1+x$ if $x \le 1$. Hence the result follows from setting $x = \frac{3\ln k}{k}$. That is,

$$k^{\frac{1}{k}} < k^{\frac{3}{k}} = e^{\frac{3\ln k}{k}} \le 1 + \frac{3\ln k}{k}.$$

□

Claim 11.2 *If $J > 10\ln r$ then $\left(\frac{2r+2J+2}{2r+J+1}\right)^{2r} \ge r$.*

Proof: First we show that $\frac{J+1}{2r+J+1} > \frac{3\ln r}{r}$ or equivalently $\frac{2r+J+1}{J+1} < \frac{r}{3\ln r}$. Now,

$$\frac{2r}{J+1}+1 \le \frac{2r}{10\ln r+1}+1 \le \frac{2r}{10\ln r}+1 \le \frac{r}{5\ln r}+1 < \frac{r}{3\ln r} \quad \text{for } r \ge 25.$$

Hence

$$\left(\frac{2r+2J+2}{2r+J+1}\right)^{2r} = \left(1+\frac{J+1}{2r+J+1}\right)^{2r} > \left(1+\frac{3\ln r}{r}\right)^{2r} > r^2 \ge r \text{ (by Claim 11.1).}$$

□

Claim 11.3 *If $J > 10\ln r$, then $\binom{2r}{r} \ge \frac{2^{2r}}{r} \ge \left(\frac{2r+J+1}{r+J+1}\right)^{2r}$.*

Proof: The first part of the inequality is straight forward and we do not give any details for that. For the second part observe that we need to show that

$$\left(\frac{2^{2r}(r+J+1)^{2r}}{r(2r+J+1)^{2r}}\right) \ge 1 \text{ or equivalently } \left(1+\frac{J+1}{2r+J+1}\right)^{2r} \ge r.$$

But the last inequality follows from the Claim 11.2.

□

Claim 11.4 $\binom{2r+J+1}{r}/\binom{2r}{r} \ge \left(\frac{2r+J+1}{r+J+1}\right)^{J+1}$.

Proof: Expanding the left hand side we obtain the following:

$$\frac{2r+1}{r+1} \times \frac{2r+2}{r+2} \times \cdots \times \frac{2r+J+1}{r+J+1}.$$

Each of these terms is at least $\frac{2r+J+1}{r+J+1}$ and there are $J+1$ terms giving us the desired inequality. □

Combining Claims 11.3 and 11.4 above we obtain that

$$\binom{2r+J+1}{r} \ge \left(\frac{2r+J+1}{r+J+1}\right)^{2r+J+1}. \tag{11.3}$$

Now we show that $\binom{t}{r}^{1/t} \geq \binom{t+1}{r}^{1/(t+1)}$, for $t = 2r + J$. But

$$\begin{aligned}
\binom{t}{r} &\geq \binom{t+1}{r}^{t/t+1} \\
\Longleftrightarrow \binom{t}{r}^{t+1} &\geq \binom{t+1}{r}^{t} \\
\Longleftrightarrow \binom{t}{r} &\geq \left(\frac{\binom{t+1}{r}}{\binom{t}{r}}\right)^{t} \\
\Longleftrightarrow \binom{t}{r} &\geq \left(\frac{t+1}{t-r+1}\right)^{t} \qquad (11.4)
\end{aligned}$$

So substituting $t = 2r + J$, we get the following:

$$\begin{aligned}
\binom{2r+J}{r} &\geq \left(\frac{2r+J+1}{r+J+1}\right)^{2r+J} \\
\Longleftrightarrow \binom{2r+J}{r}\left(\frac{2r+J+1}{r+J+1}\right) &\geq \left(\frac{2r+J+1}{r+J+1}\right)^{2r+J+1} \\
\Longleftrightarrow \binom{2r+J+1}{r} &\geq \left(\frac{2r+J+1}{r+J+1}\right)^{2r+J+1} \\
&\qquad \left(\text{since } \frac{n+1}{n+1-r}\binom{n}{r} = \binom{n+1}{r}\right).
\end{aligned}$$

But the last inequality follows from the inequality (11.3). This completes the proof of Lemma 11.3. □

Now we give an "almost exact" formula for $g(r)$; $g(r)$ is asymptotically equal to

$$2r + 2 + \left\lfloor \frac{1}{2}\ln\left(\frac{(2r+1)\pi}{2}\right) + O\left(\frac{(\ln r)^2}{r}\right)\right\rfloor.$$

So now onwards we assume that $j \leq 10\ln r$ because of the Lemma 11.3. We first consider the following equation $\binom{t+1}{r} = \left(\frac{t+1}{t+1-r}\right)^{t+1}$ and solve it asymptotically for t. We show that the soluion is $t = 2r + j + 1$ where j is some refined function of

$\ln r$.

$$\begin{aligned}
\binom{2r+j+1}{r} &= \left(\frac{2r+j+1}{r+j+1}\right)^{2r+j+1} \\
\iff \binom{2r+j+1}{r} &= \left(\frac{r+j+1}{2r+j+1}\right)^{-(2r+j+1)} \\
\iff 2^{-(2r+j+1)}\binom{2r+j+1}{r} &= \left(\frac{2r+2j+2}{2r+j+1}\right)^{-(2r+j+1)} \\
\iff 2^{-(2r+j+1)}\binom{2r+j+1}{r} &= \left(1+\frac{j+1}{2r+j+1}\right)^{-(2r+j+1)}
\end{aligned}$$

Now we use the following well known estimate on $\binom{n}{k}$ [210]:

$$\binom{n}{k} \sim \frac{2^n e^{-(n-2k)^2/2n}}{\sqrt{n\pi/2}} \quad \text{for } |n-2k| \in o(n^{3/4}).$$

Now estimating the LHS with the above estimate and taking the natural logarithm both sides we get:

$$\frac{-(j+1)^2}{2(2r+j+1)} - \frac{1}{2}\ln\left(\frac{(2r+j+1)\pi}{2}\right) \approx -(2r+j+1)\ln\left(1+\frac{j+1}{2r+j+1}\right)$$

We expand the RHS using the expansion $\ln(1+x) = (x - x^2/2 + x^3/3 - \cdots)$ and ignoring the second order term since for large values of r, $(\ln r)^2/r$ is negligible. Thus we get the following:

$$J = j+1 \sim \frac{1}{2}\ln\left(\frac{(2r+j+1)\pi}{2}\right) + O\left(\frac{(\ln r)^2}{r}\right).$$

Now because of the inequalities above, it is easy to observe that

$$\begin{aligned}
\binom{t+1}{r} &\geq \left(\frac{t+1}{t+1-r}\right)^{t+1} \quad \text{for } t \geq 2r+J \text{ and} \\
\binom{t+1}{r} &< \left(\frac{t+1}{t+1-r}\right)^{t+1} \quad \text{for } t < 2r+J.
\end{aligned} \tag{11.5}$$

Now we show that

$$\begin{aligned}\binom{t}{r}^{1/t} &\geq \binom{t+1}{r}^{1/(t+1)} \text{ if } t \geq 2r+J \text{ and}\\ \binom{t}{r}^{1/t} &< \binom{t+1}{r}^{1/(t+1)} \text{ if } t < 2r+J.\end{aligned}$$

But because of the inequality (11.4), it is equivalent to inequality (11.5), which we have already shown. But t is a integer and hence it takes optimum value when we have $t = 2r + \lfloor J \rfloor$. So in our context we get the following value for $g(r)$,

$$2r + 2 + \left\lfloor \frac{1}{2} \ln \left(\frac{(2r+1)\pi}{2} \right) + O\left(\frac{(\ln r)^2}{r} \right) \right\rfloor$$

or the value for which $\binom{g(r)}{r+1}^{1/g(r)}$ is maximized. This completes the proof of Lemma 11.2. □

Lemma 11.2 gives us the following theorem.

Theorem 11.9 *For a fixed r, $\exists$ graphs G such that $\mathscr{M}_r(n)$ is at least $\binom{g(r)}{r+1}^{n/g(r)}$, where $g(r)$ is of Lemma 11.2, that is,*

$$g(r) = \begin{cases} 2r+3 & 0 \leq r \leq 11 \\ 2r+4 & 12 \leq r \leq 100 \\ 2r+2+\left\lfloor \frac{1}{2}\ln\left(\frac{(2r+1)\pi}{2}\right) + O\left(\frac{(\ln r)^2}{r}\right)\right\rfloor & r > 100. \end{cases}$$

For a fixed r, let lb_r and ub_r denote a base of exponent in lower bound and upper bound on $\mathscr{M}_r(n)$, i.e., $lb_r^n \leq \mathscr{M}_r(n) \leq ub_r^n$. When $r \geq 3$, we obtain tighter upper bounds on $\mathscr{M}_r$ by directly finding the roots of the polynomials coming from the recurrences in MAX-r-RIS algorithm, as we did to obtain values in Table 11.1. We can see that the upper bound obtained this way and the lower bound coming from Lemma 11.2 are already very close, as Table 11.2 shows.

For small values of r, these upper bounds could be made equal to the lower bound by choosing α and β appropriately in the definition of μ and by doing the analysis similar to the one in Theorem 11.8. For an example, when $r = 3$ we can take $\alpha = 1.73$ and $\beta = 2$ and show that $lb_3 = ub_3$. We do not go into the details for the the upper bounds for larger values of r as (a) they require a lot of case

r	3	4	5	8	10	15
lb_r	1.71149	1.7468	1.7734	1.8253	1.8474	1.8828
ub_r	1.73615	1.76357	1.78554	1.83111	1.85136	1.88452
$ub_r - lb_r$	0.02466	0.016782	0.012131	0.0057618	0.0039415	0.0017377

Table 11.2: Bounds on the Number of Maximal-r-Regular Induced Subgraphs for Small Values of r.

distinctions and (b) the upper bounds we have are already too close to the lower bounds and they increasingly become closer for larger values of r.

11.3 Improved Algorithms for $r = 1$ and 2

Our generic algorithm **Max-r-RIS** finds a maximum r-regular induced subgraph in time $\mathcal{O}(1.6957^n)$ and $\mathcal{O}(1.7069^n)$ for $r = 1$ and 2 respectively. Our algorithmic upper bound proofs (on $\mathscr{M}_r(n)$) of Section 11.2 enumerates all maximal r-regular subgraphs in time $\mathcal{O}(1.58469^n)$ and $\mathcal{O}(1.66181^n)$ for $r = 1$ and 2 respectively. These algorithms immediately give better runtime bound than those in Section 11.1 for finding a maximum r-regular induced subgraph for $r = 1$ and 2. Here we further improve these algorithms.

11.3.1 Maximum Induced Matching (MIM)

Given a graph $G = (V, E)$, we get our improvement from the following observations. Let v be a vertex having a neighbor u such that $N(u) \subseteq N[v]$. Consider the set $\mathcal{M}_v$ of maximum sized induced matching having v (these matchings may not be the maximum sized induced matching of G). Then we have the following simple lemma.

Lemma 11.4 *Let G be a graph and v be a vertex and $u \in N(v)$ such that $N(u) \subseteq N[v]$ then there exists a $M' \in \mathcal{M}_v$ such that it contains the edge (v, u).*

Proof: Suppose there does not exist any matching in $\mathcal{M}_v$ containing the edge (v, u). Let M be any matching in $\mathcal{M}_v$ and $(v, x) \in M$, $x \neq u$. Then $M' = M - (v, x) + (v, u)$ is an induced matching containing v since $N(u) \subseteq N[v]$ and of size $|M|$, a contradiction. □

The other observation relates MIM of G to Maximum Independent Set (MIS) of square of the *line graph* of G. The *line graph*, $L(G)$ of $G = (V, E)$ is the graph whose vertices are edges of G, and two edges e_1, e_2 are adjacent if and only if they are adjacent edges in G. G^i (ith power of G) is a graph on V with edges between two vertices v_1 and v_2 if and only if there is a path of length at most i between v_1 and v_2 in G.

Lemma 11.5 ([46]) *Let G be a graph then* $MIM(G) = MIS(L(G)^2)$.

So our algorithm uses branching on a vertex v when the maximum degree of the graph is at least 5 and distinguishes cases based on Lemma 11.4. When the maximum degree of the graph is at most 4, we use the well known algorithms to find a maximum independent set [116, 208] in $L(G)^2$.

Our detailed algorithm for Maximum Induced Matching is depicted in Figure 11.3.

The correctness of the algorithm follows from Lemmas 11.4 and 11.5. Observe that once we have found an edge for a possible MIM, all the neighbors of its endpoints are removed from the graph as the edges emanating from these vertices can no longer be part of the MIM. Observe that when branching steps take place, v has degree $d \geq 5$. Hence if there exists a $u \in N(v)$ such that $N(u) \subseteq N[v]$ then the branching step of the algorithm is dominated by the following recurrence:

$$T(n) \leq T(n-1) + T(n-6).$$

In the other branching step of the algorithm, every vertex $u \in N(v)$ has at least one neighbor x such that $x \notin N[v]$. Hence in this branching step n goes down by $n - (d+2)$, if the degree of v is d. So we get the following recurrence:

$$T(n) \leq \max_{d \geq 5} \Big\{ T(n-1) + dT(n-d-2) \Big\}.$$

The solutions to these recurrences are given by the positive roots of the polynomials $\{x^{d+2} - x^{d+1} - d \mid d \geq 5\}$. The positive root of $x^7 - x^6 - 5$ is the largest and its value is 1.47856. If the maximum degree of the graph is at most 4, then we find a maximum independent set of the square of the line graph $L(G)$ of G. We could use the best known polynomial space algorithm, developed by Fomin et al. [116],

```
Algorithm MaxIM ($G = (V, E), M$)
Input: A graph $G = (V, E)$ and $M$ contains the set of edges of induced
       matching.
Output: A set $M$ consisting of edges of a maximum induced matching.
  (Preprocessing Step)
  remove all vertices of degree 0 recursively.
  (Branching Steps)
  choose a vertex $v \in V$ of maximum degree
  if $deg(v) \geq 5$ then
      $M_1 \leftarrow$ MaxIM$(G - v, M)$
      if $\exists u \in N(v)$ such that $N(u) \subseteq N[v]$ then
          $M_2 \leftarrow$ MaxIM$(G - N[v] - N[u], M \cup \{v, u\})$
      else
          $M_2 \leftarrow M_1$
          for $u \in N(v)$ do
              $M_u \leftarrow$ MaxIM$(G - N[v] - N[u], M \cup \{v, u\})$
              if $|M_u| > |M_2|$ then $M_2 \leftarrow M_u$
      return the set of maximum size among $\{M_1, M_2\}$
  else
      (Maximum Independent Set Step)
      Construct the line graph $L$ of $G$ and obtain the MIS $I$ of $L(G)^2$ and
      return $I$.
```

Figure 11.3: An Algorithm to Find a Maximum Induced Matching of a Graph

to find a MIS, as a subroutine in our algorithm or the best known exact algorithm to find MIS developed by Robson [208], taking exponential space.

Proposition 11.1 ([116],[208]) *Given a graph $G = (V, E)$ on n vertices,* Maximum Independent Set *can be found in* (a) $O^*(1.2210^n)$ *time and space polynomial in n and* (b) $O^*(1.2108^n)$ *time and space exponential in n.*

Since the maximum degree of G is at most 4 when we apply the MIS algorithm, the number of edges is bounded by $2n$ and hence the number of vertices in $L(G)^2$ is at most $2n$. So the MIS step takes either $(1.221)^{2n} = (1.4904^n)$ or $(1.2108)^{2n} = (1.46604^n)$ depending on which MIS algorithm we use as a subroutine. All these put together show that the Maximum Induced Matching problem can be solved in (a) $\mathcal{O}(1.4904^n)$ time and space polynomial in n or in (b) $\mathcal{O}(1.4786^n)$ time and space exponential in n. Thus we have

Theorem 11.10 *Let $G = (V, E)$ be a graph on n vertices, then a* MIM *can be found in (a) $\mathcal{O}(1.4904^n)$ time and space polynomial in n or in (b) $\mathcal{O}(1.4786^n)$ time and space exponential in n.*

A problem which is a generalization of the Maximum Induced Matching is δ-Separating Maximum Matching (δ-SepMM). The problem is defined as follows:

> δ-SepMM [216]: Given a graph $G = (V, E)$ and a positive integer δ, find a maximum sized $M \subseteq E$ such that the distance between any two edges is at least δ. The *distance* between edges e and e' is the length of the shortest path between the vertex corresponding to e and the vertex corresponding to e' in the line graph $L(G)$ of G.

Stockmeyer and Vazirani [216] showed that this problem is NP-complete for any fixed integer $\delta \geq 2$. For $\delta = 2$, it is precisely Maximum Induced Matching. δ-Separating Maximum Matching problem can also be solved in the way we have solved Maximum Induced Matching problem with slight modification like noting that

$$\delta - SepMM(G) = MIS(L(G))^{\delta}.$$

The algorithm is exactly as in Figure 11.3 except that when we include an edge in a possible δ-separating maximum matching then we remove all the neighbors of the end points of this edge which are at distance at most $\delta - 1$ and then we use an algorithm to find MIS in $(L(G))^{\delta}$. Thus we obtain this.

Theorem 11.11 *Let $G = (V, E)$ be a graph on n vertices and $\delta \geq 2$ be a positive integer, then the δ-*Separating Maximum Matching *problem can be solved in (a) $\mathcal{O}(1.4904^n)$ time and space polynomial in n or in (b) $\mathcal{O}(1.4786^n)$ time and space exponential in n.*

11.3.2 Maximum-2-Regular Induced Subgraph

In this section we obtain an improved algorithm for M-2-RIS by using the measure μ defined in Section 11.2 and by refining a few branching rules. Consider a vertex v with its neighbor set $N(v)$ partitioned into W_1, W_2 and W_3 as in the proof of Theorem 11.8, such that every vertex $u \in W_1$ has $N(u) \subseteq N[v]$, every vertex in

W_2 has a unique neighbor x such that $x \notin N[v]$ while every vertex $u \in W_3$ has at least 2 neighbors not in $N[v]$. Now we obtain a following easy lemma which helps in refining a branching step.

Lemma 11.6 *Let $G = (V, E)$ be a graph and v be a vertex with its neighbor set $N(v)$ partitioned into W_1, W_2 and W_3 as defined above. Then a maximum sized solution containing v and two vertices from W_1 (this solution may not be an optimal solution of G) can be obtained by taking any pair of vertices from W_1 having an edge between them.*

Proof: Observe that any optimal solution containing v and two of its neighbors x and y from W_1, has an induced triangle containing v, x, and y. We can obtain another optimal solution by replacing this triangle with any other triangle formed with v and two vertices from W_1 having an edge between them. □

Assume that we have a vertex v and its neighbor set $N(v)$ partitioned into W_1, W_2 and W_3 as above and we are considering a branching on v (that is excluding v or including in R v and two of its neighbors in R). In the proof of Theorem 11.8, the cases which lead to the worst case running time are

(a) when W_1 is large $(4 \leq |W_1| \leq 6)$ and

(b) when $2 \leq$ maximum degree $d \leq 6$,

$$W_1 = W_2 = \emptyset \text{ and } \forall (y, z) \in W_3 \times W_3,\ (y, z) \notin E.$$

We refine these cases and obtain an improved algorithm to find a maximum 2-regular induced subgraph. Lemma 11.6 refines the first case as we only need to consider one branch for the vertices of W_1. We refine the second case by refining the branches when we include v and its two neighbors in R as follows: suppose we have $(y, z) \in W_3 \times W_3$ and $|S^v_y \cup S^v_z = \{w_1, w_2\}| = 2$ then either $N(w_1) \cup N(w_2) \subseteq N[v]$ or $N(w_1) \cup N(w_2) \nsubseteq N[v]$. In the first we can take either w_1 or w_2 and make a 2-regular induced subgraph consisting of $\{v, y, z, w_1\}$, leading to a decrease in μ by $\beta(d+1) - 2\beta$. Let $p \in (N(w_1) \cup N(w_2)) - N[v]$ and p be a neighbor of w_1. Then in this case after applying the branches where we include v and two of its neighbors, branch on w_2; that is either $w_1 \notin R$ or $w_1 \in R$. In the first case $w_1 \notin R$ decreases μ by $\beta(d + 1) - 2\beta$ as w_2 becomes a unique neighbor of y and z, leading to an inclusion in R. Similarly the second case when $w_1 \in R$ decreases μ by $\beta(d+1)-3\beta$ as $\{v, y, z, w_1\}$ forms an induced cycle. The worst case for $(y, z) \in W_3 \times W_3$ is when $|S_y^v \cup S_z^v| = 3$. Hence a typical recurrence which guides the running time of the algorithm is given by:

$$\begin{aligned} f(\mu) \leq\ & f(\mu-\beta) + f(\mu-\beta(d+1)) + |W_1||W_2|f(\mu-\beta(d+2)) \\ & +|W_1||W_3|f(\mu-\beta(d+3)) + \binom{|W_2|}{2} f(\mu-\beta(d+2)) \\ & +|W_2||W_3|f(\mu-\beta(d+2)-2\delta) + \binom{|W_3|}{2} f(\mu-\beta(d+1)-3\delta). \end{aligned}$$

We take $\alpha = 0.758$ and $\beta = 1$. This gives us the following theorem using computation with MATLAB.

Theorem 11.12 *Let $G = (V, E)$ be a graph on n vertices, then the* Maximum 2-Regular Induced Subgraph *problem can be solved in $\mathcal{O}(1.62355^n)$ time.*

11.4 Conclusion

In this chapter we developed an $\mathcal{O}(c^n)$ time exact algorithms for Maximum r-Regular Induced Subgraph problems for any fixed constant r, where $c < 2$ is a constant depending on r alone. We also showed that if r is a constant then the number of maximal r-regular induced subgraphs on a graph on n vertices is bounded by $o(2^n)$. Then we gave very tight lower and upper bounds on the number of maximal r-regular induced subgraphs on n vertices. Most of our algorithms were simple to describe but their analyzes were non-trivial. We analyzed recurrences having binomial coefficients and believe that these may trigger some new results in the area of exact algorithms. Finally, we used the results obtained on the enumeration version of Maximum r-Regular Induced Subgraph problems to give a non trivial exact algorithm for Induced r-Regular Subgraph Isomorphism when r is a constant. The other problems for which we can give non trivial exact algorithms based on the algorithms and the techniques developed in this chapter include Efficient Edge Dominating Set [129], and δ-Separating Maximum Matching [216] problems.

It will be interesting to find other applications of the algorithms developed in this chapter. Finding a non trivial exact algorithm for Induced Subgraph Isomorphism problem, even for special classes of H, remains open. Here we obtained an efficient algorithm for Induced Subgraph Isomorphism when H is a r-regular graph for a constant r.

Part IV

Conclusion and Future Directions

12 Summary and Future Research

In this final chapter we summarize the results presented in this thesis and look for possible directions to move ahead.

12.1 Undirected Feedback Vertex Set

We proved that graphs with minimum degree 3 having a small fvs possess short cycles. Using this we obtained faster algorithms for parameterized feedback vertex set problem on undirected graphs. Our main result achieves a significant improvement in the dependence on k (the parameter) of the running time. We get an algorithm with $O\left(\left(\frac{12\log k}{\log\log k}+6\right)^k n^\omega\right)$ running time.

A number of advances have been made on reducing the $f(k)$ for the FVS problem. Dehne et. al. [69] have obtained an algorithm for FVS problem that runs in time $O(c^k n^3)$, where $c = 10.567$. Independently, Guo et. al.[138] have also obtained a $O(c^k mn)$, where $c = 37.7$, time algorithm for the FVS problem.

Most of the recent improvements on designing FPT algorithms are based (in part) directly or indirectly on the ideas used in this paper. These papers uses recently developed *iterative compression* technique which is a useful technique for designing FPT algorithms for minimization problems. Both these algorithms obtain a linear size kernel in the *compression* step which is central to the analysis of the new algorithms. In combination with Lemma 4 of [69], a kernel of size $8k$ can directly be obtained as a consequence of Lemma 3.9 (obtained in this thesis) for the compression step which itself will give a better $O(c^k n^{O(1)})$, c a constant, algorithm for fvs than presented in [138]. In fact authors in [69] have proved a relaxed version of Lemma 3.9 obtained here. They modify the definition of matching on degree two vertices by allowing any arbitrary length path between them unlike us who wants it be adjacent. More precisely, they define $M' = \{(x, y) | deg_T(x) = deg_T(y) = 2$ and there is a path between x and y in T. $\}$ and by $M \subseteq M'$ they mean set of paths whose pairwise intersection is empty. With, this modified definition of M they prove a stronger bound of $N/2$ on W (W as defined in Lemma 3.9) and obtain a kernel of size $4k$ in the compression step which leads to an improved c^k algorithm for fvs. Compression lemmas obtained in [69] and [138] are similar in flavor to our Lemma 3.9. Recently, combining branching and iterative compression Chen et. al. [52] gave $O(5^k n^{O(1)})$ time algorithm for FVS problem. It will be interesting to improve the time complexity of this algorithm.

Apart from its application to the design of FPT algorithms, our Lemma 3.9 and Theorems 3.2 and 3.5 may be of independent interest in extremal graph theory. One of the interesting question here is whether the bound on the girth in Theorem 3.5 is optimal.

12.2 Directed Feedback Vertex Set

Here we have obtained efficient algorithms for parameterized feedback arc and vertex set problem on weighted tournaments. For the feedback arc set problem, the complexity of the algorithms in unweighted and weighted (with weights at least 1) versions are the same while this is not the case for the feedback vertex set problem.

We have also given FPT algorithms for the parametric duals of directed feedback vertex and arc set problems in oriented directed graphs and directed graphs respectively. Dual of directed feedback vertex set problem in directed graphs is shown to be $W[1]$-hard. In line with parameterizing above the guaranteed values, the parameterized complexity of the following questions are also interesting and remains open.

- Given an oriented directed graph on n vertices, does it have a subset of at least $\lfloor \lg n \rfloor + k$ vertices that induces an acyclic subgraph?
- Given an oriented directed graph on n vertices and m arcs, does it have a subset of at least $m/2 + 1/2(\lceil n - 1/2 \rceil) + k$ arcs that induces an acyclic

subgraph ?

- Given a directed graph on n vertices and m arcs, does it have a subset of at least $m/2 + k$ arcs that induces an acyclic subgraph ?

In a recent development DFVS problem has been shown to be FPT in general directed graphs. The time complexity of this algorithm is $O(k!8^k n^{O(1)})$ [54]. In this diercetion one can ask following questions :

- Can we get $O(c^k n^{O(1)})$ time algorithm for DFVS in general directed graphs?
- Can we get polynomial sized kernel for DFVS in polynomail time ?

12.3 FPT Characterization for Problems in Graphs with no Small Cycles

We showed that if the input graphs do not possess short cycles then the neighborhood problems like dominating set, independent set and their variants are fixed parameter tractable. We have also shown that the restriction on girth is optimal if we do not put further restriction on the graph classes. This is the first time, to our knowledge, the complexity of graph problems are classified by girth.

Most of the algorithms given here are just parameterized complexity classification algorithms. We believe that the vast literature known for these problems can be applied to obtain more efficient FPT algorithms. Obtaining a $O(c^k n^{O(1)})$, c a constant, algorithm for all these problems remain an open problem.

We also gave an improved approximation algorithm for dominating set problem in graphs without cycles of length 3 or 4. It would be interesting to explore such better approximation algorithms for problems on graphs with no small cycles.

12.4 Directed Maximum Leaf Problems

Research initiated by results in this thesis was continued by Bonsma and Dorn who proved in [37] that every strongly connected digraph of order n with minimum in-degree at least 3 has a out-branching with at least $\sqrt{n}/4$ leaves. Thus, the maximum guaranteed number $\lambda(n)$ of leaves in a strongly connected digraph of order n with minimum in-degree at least 3 is $\Theta(\sqrt{n})$. It would be interesting to obtain the maximum constant c such that $\lambda(n) \geq c\sqrt{n}$.

Using several ideas of this thesis, some new ideas and treewidth rather than pathwidth, Bonsma and Dorn [37] designed algorithms of complexity $2^{O(k \log k)}n^{O(1)}$ for both k-DMLOT and k-DMLOB. Using another approach, Kneis, Langer and Rossmanith [166] obtained an $4^k n^{O(1)}$ time algorithm for k-DMLOB. It is not difficult to see that this algorithm implies an $4^k n^{O(1)}$ time algorithm for k-DMLOT.

We conclude by pointing out that in a recent paper [84], Drescher and Vetta describe an $O(\sqrt{\text{OPT}})$-approximation algorithms for DMLOB, where OPT is the maximum number of leaves in an out-branching of the input digraph.

12.5 Exact Algorithms

In this thesis we first developed a technique by which we can use a parameterized algorithm of time complexity $O^*((4-\epsilon)^k)$, where k is the parameter and $\epsilon > 0$, to obtain an exact algorithm of time complexity $O^*((2-\eta)^n)$, $\eta > 0$ and then applied the technique to various problems. This technique is based on a careful use of the parameterized algorithm for certain values of the parameter and brute-force for other values.

Algorithms for enumerating maximal independent sets have been used in most of the algorithms developed for Coloring problem. Here, we further showed the power of this technique by obtaining significantly improved and the best known exact algorithms for minimum odd cycle transversal, maximum k-colorable induced subgraph for small values of k, maximum split graphs, minimum maximal matching and minimum edge dominating set. Though all our algorithms have enumeration of maximal independent set as a subroutine, we had to transform the problem or use interesting characterization to use the MIS enumeration algorithm.

Branching and dynamic programming on graphs of bounded treewidth are very powerful techniques to design efficient exact algorithms. In this chapter, we combined these two techniques in different ways and obtained improved exact algorithms for #3-COLORING, MMM and its variants. We also applied the technique to design fixed parameter tractable algorithms and obtained fast algorithms for k-WVC and k-WEDS which also shows the versatility of our technique. The most important aspects of this technique are that the resulting algorithms are very elegant and simple while at the same time the analysis of these algorithms is non-trivial.

It would be interesting to find some other applications of the techniques presented here in the design of exact exponential time algorithms and fixed parameter tractable algorithms. It would be also nice to see other applications of the algorithm for enumerating maximal independent sets. Some of the concrete open problems are:

- Improved exact algorithms for DOMINATING SET, at least in bipartite graphs using enumeration of maximal independent sets or otherwise.
- Improved exact algorithms for HAMILTONIAN CYCLE in special graph classes like bipartite graphs or split graphs.

12.6 Maximum r-Regular Induced Subgraph Problems

We developed an $\mathcal{O}(c^n)$ time exact algorithms for MAXIMUM r-REGULAR INDUCED SUBGRAPH problems for any fixed constant r, where $c < 2$ is a constant depending on r alone. We also showed that if r is a constant then the number of maximal r-regular induced subgraphs on a graph on n vertices is bounded by $o(2^n)$. Then we gave very tight lower and upper bounds on the number of maximal r-regular induced subgraphs on n vertices. All our algorithms were simple to describe but their analyzes were non-trivial and involved a different measure than the usual number of vertices to measure the progress of the algorithms. We analyzed recurrences having binomial coefficients and believe that these may trigger some new results in the area of exact algorithms. Finally, we used the results obtained on the enumeration version of MAXIMUM r-REGULAR INDUCED SUBGRAPH problems to give a non trivial exact algorithm for INDUCED r-REGULAR SUBGRAPH ISOMORPHISM when r is a constant. The other problems for which we can give non trivial exact algorithms based on the algorithms and the techniques developed in this chapter include EFFICIENT EDGE DOMINATING SET [129], δ-SEPARATING

Maximum Matching [216] and Maximum Bounded Degree Induced Subgraph problems.

It will be interesting to find other applications of the algorithms developed in this chapter. Finding a non trivial exact algorithm for Induced Subgraph Isomorphism problem, even for special classes of H, remains open. Here we obtained an efficient algorithm for Induced Subgraph Isomorphism when H is a r-regular graph for a constant r.

12.7 Conclusions

Parameterized complexity has emerged as an important practical direction to pursue for problems where a small range of parameter values is of particular interest[33, 83, 32, 103]. The study of parameterized complexity and exact exponential time algorithms has given rise to novel and interesting algorithmic techniques to solve some difficult problems exactly. We conclude this thesis with specific two open problems which is not mentioned earlier.

1. Does Odd Cycle Transversal problem parameterized by solution size, k, admit polynomial size kernel?

2. Can we get an $O(c^n)$ time algorithm for Subgraph Isomorphism problem?

I will end this thesis with four lines of a poem by Robert Frost which sums up all.

Woods are lovely, dark and deep.

But I have promises to keep,
And miles to go before I sleep,
And miles to go before I sleep.

13 Publications

List of Papers used for the thesis.

1. Spanning Directed Tree with many Leaves (with N. Alon, F. V. Fomin, G. Gutin and M. Krivelevich). Accepted to appear in SIAM Journal on Discrete Mathematics.

   Preliminary versions of the paper appeared with titles 'Parameterized Algorithms for Directed Maximum Leaf Problems ' and 'Better Algorithms and Bounds for Directed Maximum Leaf Problems ' in the proceedings of ICALP 2007 and FSTTCS 2007 respectively.

2. On Two Techniques of Combining Branching and Treewidth (with F. V. Fomin, S. Gaspers, and A. A. Stepanov). Accepted for Publication in 'Algorithmica', Springer Verlag. Invited paper for ISAAC 2006. (Accepted in May 2007).

   A preliminary version of the paper appeared with title 'Branching and Treewidth Based Exact Algorithms' in the proceedings of ISAAC 2006.

3. Short Cycles make W-hard problems hard: FPT algorithms for W-hard problems in Graphs with no short cycles (with V. Raman). Algorithmica. Volume 52, Issue 2, Pages 203-225 (2008).

   A preliminary version of the paper appeared with title 'Triangles, 4-cycles and Parameterized (In-) Tractability' in the proceedings of SWAT 2006.

4. On the Complexity of Some Colorful Problems Parameterized by Treewidth (with M. Fellows, F. V. Fomin, D. Lokshtanov, F. Rosamond, S. Szeider and

C. Thomassen). In the proceedings of COCOA'07: (Springer Verlag, LNCS **4616**) 366-377 . (Invited Paper.)

5. Improved Fixed Parameter Tractable Algorithms for Two Edge Problems – MAXCUT and MAXDAG (with V. Raman). Information Processing Letters (IPL). Volume 104(2), Pages 65-72 (2007).

6. Efficient Exact Algorithms through Enumerating Maximal Independent Sets and Other Techniques (with V. Raman and S. Sikdar). Theory of Computing Systems. Volume 41, Issue 3, Pages 563-587 (2007).

   A preliminary version of the paper appeared with title 'Improved Exact Exponential Algorithms for Vertex Bipartization and Other Problems ' in the proceedings of ICTCS 2005.

7. Faster Fixed Parameter Tractable Algorithms for Finding Feedback Vertex Sets (with V. Raman and C. R. Subramanian). ACM Transactions on Algorithms (TALG). Volume 2, Issue 3, Pages 403-415 (2006).

   Preliminary versions of the paper appeared with title 'Improved Fixed Parameter Tractable Algorithms for the Undirected Feedback Vertex Set problem' and 'Faster Algorithms for Feedback Vertex Set' in the proceedings of ISAAC 2002 and GRACO 2005 respectively.

8. Parameterized Algorithms for Feedback Set Problems and Their Duals in Tournaments (with V. Raman). Theoretical Computer Science (TCS). Volume 351, Issue 3, Pages 446-458 (2006).

   Preliminary versions of the paper appeared with title 'Parameterized Complexity of Directed Feedback Set Problems in Tournaments' and 'Improved Parameterized Algorithms for Feedback Set Problems in Weighted Tournaments' in the proceedings of WADS 2003 and IWPEC 2004 respectively.

9. Improved Exact Algorithms for Counting 3- and 4- Colorings (with F. V. Fomin and S. Gaspers) In the proceedings of COCOON'07: (Springer Verlag, LNCS **4598**) 65-74.

10. Fast Exponential Algorithms for Maximum r-Regular Induced Subgraph Problems (with S. Gupta and V. Raman). In the proceedings of FSTTSC '06: (Springer Verlag, LNCS **4337**) 139-151.

Bibliography

[1] E. Aarts and J. K. Lenstra, editors, *Local search in combinatorial optimization.* Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons Ltd., Chichester, 1997. A Wiley-Interscience Publication.

[2] J. Alber, H. L. Bodlaender, H. Fernau and R. Niedermeier, *Fixed Parameter Algorithms for Dominating Set and Related Problems on Planar Graphs. Algorithmica* **33** (2002) 461-493.

[3] J. Alber, H. Fan, M. R. Fellows, H. Fernau, R. Niedermeier, F. Rosamand and U. Stege, *Refined Search Tree Techniques for Dominating Set on Planar Graphs.* In the proceedings of 26th International Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Computer Science **2136** (2001) 111-122.

[4] J. Alber, H. Fan, M. R. Fellows, H. Fernau, R. Niedermeier, F. Rosamond and U. Stege, *A refined search tree technique for Dominating Set on planar graphs.* Journal of Computer and System Sciences 71(4): (2005) 385-405.

[5] J. Alber, M. R. Fellows and R. Niedermeier, *Polynomial time data reduction for dominating Set.* Journal of the ACM **51(3)** (2004) 363-384.

[6] V. E. Alekseev, *On easy and hard hereditary classes of graphs with respect to the Independent Set problem.* Discrete Applied Mathematics **132(1-3)** (2003) 17-26.

[7] V. E. Alekseev, D. V. Korobitsyn and V. V. Lozin, *Boundary classes of graphs for the Dominating Set problem.* Discrete Mathematics **285(1-3)** (2004) 1-6.

[8] N. Alon, *Ranking Tournaments.* Siam Journal on Discrete Mathematics, 20(1), (2006) 137-142.

[9] N.Alon, S.Hoory and N.Linial, *The Moore Bound for Irregular Graphs.* Graphs and Combinatorics **18** (2002) 53-57.

[10] N. Alon and J. Spencer, *The Probabilistic Method.* Wiley, NY, 2nd Ed., 2000.

[11] N. Alon, R. Yuster and U. Zwick, *Color-Coding.* Journal of the Association for Computing Machinery, **42**(4) (1995) 844-856.

[12] O. Angelsmark and P. Jonsson, *Improved Algorithms for Counting Solutions in Constraint Satisfaction Problems.* In the proceedings of the 9th International Conference on Principles and Practice of Constraint Programming, (2003) 81-95.

[13] S. Arnborg, *Efficient algorithms for combinatorial problems on graphs with bounded decomposability - a survey. BIT* **25** (1985) 2-23.

[14] S. Arnborg, J. Lagergren and D. Seese, *Easy problems'for tree-decomposable graphs.* Journal of Algorithms **12** (1991) 308-340.

[15] S. Arnborg and A. Proskurowski, *Linear Time Algorithms for NP-hard problems restricted to partial k-trees.* Discrete Applied Mathematics **23** (1989) 11-24.

[16] L. Babai, W. M. Kantor and E. M. Luks, *Computational Complexity and the Classification of Finite Simple Groups.* In the Proceedings of FOCS'83. (1983) 162-171.

[17] R. Balasubramanian, M.R. Fellows and Venkatesh Raman, *An Improved Fixed Parameter Algorithm for Vertex Cover.* Information Processing Letter **65** (1998) 163-168.

[18] J. Bang-Jensen and G. Gutin, *Digraphs Theory, Algorithms and Applications*, (2001) Springer-Verlag.

[19] N. Bansal and V. Raman, *Upper Bounds for MAX-SAT further improved.* In the proceedings of 10th International Symposium on Algorithms and Computation, Lecture Notes in Computer Science **1741** (1999) 247-258.

[20] R. Bar-Yehuda, D. Geiger, J. Naor, R. M. Roth, *Approximation Algorithms for the Feedback Vertex Set Problem with Applications to Constraint Satisfaction and Bayesian Inference.* In the proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms, (1994) 344-354.

[21] A Becker, R. Bar-Yehuda and D. Geiger, *Random Algorithms for the Loop Cutset Problem.* Journal of Artificial Intelligence Research **12** (2000) 219-234.

[22] R. Beigel, *Finding maximum independent sets in sparse and general graphs.* In the proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms (SODA 1999), ACM and SIAM, (1999) 856-857.

[23] J. C. Bermond, A. Germa, M. C.Heydemann and D. Sotteau, *Girth in Digraphs.* Journal of Graph Theory, **4** (3) (1980) 337-341.

[24]) D. Bienstock, N. Robertson, P. D. Seymour and R. Thomas, *Quickly excluding a forest.* Journal of Combinatorial Theory, Series B 52(2): (1991) 274-283.

[25] A. Björklund and T. Husfeldt, *Exact Algorithms for Exact Satisfiability and Number of Perfect Matchings.* In the proceedings of International Colloquium on Automata, Languages and Programming (ICALP). LNCS 4051: (2006) 548-559.

[26] M. Bläser, *Computing small partial coverings.* Information Processing Letters **85(6)** (2003) 327-331.

[27] H.L. Bodlaender, *On linear time minor tests and depth-first search.* Journal of Algorithms 14 (1993), 1-23.

[28] H. L. Bodlaender, *A tourist guide through treewidth.* Acta Cybernetica, 11 (1993) 1-21.

[29] H.L. Bodlaender, *A Linear-Time Algorithm for Finding Tree-Decompositions of Small Treewidth.* SIAM Journal on Computing 25 (1996), 1305-1317.

[30] H. L.Bodlaender, *Dynamic programming algorithms on graphs with bounded tree-width.* In the proceedings of the 15th International Colloquium on Automata, Languages and Programming (ICALP), Lecture Notes in Computer Science **317** (1988) 105-117.

[31] H. L. Bodlaender, *A partial k-arboretum of graphs with bounded treewidth.* Theoretical Computer Science, 209 (1998), 1-45.

[32] H. L. Bodlaender, R. G. Downey, M. R. Fellows, and H. T. Wareham, *The Parameterized complexity of sequence alignment and consensus.* Theoretical Computer Science **147** (1995) 31-54.

[33] H. L. Bodlaender, M. R. Fellows and T. Warnow, *Two Strikes Against Perfect Phylogeny.* In the proceedings of the 19th International Colloquium on Automata, Languages and Programming (ICALP 92), Lecture Notes in Computer Science, **623** (1992) 273-283.

[34] H. L. Bodlaender and F. V. Fomin, *Equitable colorings of bounded treewidth graphs.* Theoretical Computer Science 349, (2005) 22-30.

[35] V. Bonifaci, U. D. Iorio and L. Laura, *On the Complexity of Uniformly Mixed Nash Equilibria and Related Regular Subgraph Problems.* In the proceedings of FCT'05. LNCS 3623: (2005) 197-208.

[36] P.S. Bonsma, T. Brueggermann and G.J. Woeginger, *A faster FPT algorithm for finding spanning trees with many leaves.* Lecture Notes in Computer Science, **2747** (2003) 259-268.

[37] P.S. Bonsma and F. Dorn, Tight bounds and faster algorithms for Directed Max-Leaf. Lecture Notes in Computer Science, **5193** (2009) 222-233.

[38] D. P. Bovet and P. Crescenzi, *Introduction to the Theory of Complexity'*, Prentice-Hall, New York (1993).

[39] A. Brandstädt, *Partitions of Graphs into one or two Independent Sets and Cliques.* Discrete Mathematics 152 (1-3): (1996) 47-54.

[40] J. M. Byskov, *Exact Algorithms for Graph Colouring and Exact Satisfiability.* PhD Dissertation, (2004).

[41] J. M. Byskov, *Enumerating Maximal Independent Sets with Applications to Graph Colouring.* . Operations Research Letters 32(6): (2004) 547-556.

[42] L. Cai and J. Chen, *On Fixed-Parameter Tractability and Approximation of NP Optimization Problems.* Journal of Computer and System Sciences, **54** (1997) 465-474.

[43] L.Cai, J.Chen, R.Downey and M.Fellows, *The parameterized complexity of short computations and factorization.* University of Victoria, Technical Report, Department of Computer Science, July, 1993.

[44] L.Cai, J.Chen, R.Downey and M.Fellows, *Advice Classes of Parameterized Tractability.* Annals of Pure and Applied Logic, **84** (1997) 119-138.

[45] L. Cai and D. Juedes, *On the Existence of Subexponential Parameterized Algorithms.* Journal of Computer and System Sciences, **67** (4) (2003) 789-807.

[46] K. Cameron, *Induced Matchings.* Discrete Applied Mathematics 24: 97-102 (1989).

[47] D. M. Cardoso, M. Kaminski and V. Lozin, *Maximum k-Regular Induced Subgraphs.* Rutcor Research Report (RRR) 3, (2006).

[48] M. Cesati, Compendium of parameterized problems, Sept. 2006. `http://bravo.ce.uniroma2.it/home/cesati/research/compendium.pdf`

[49] L. S. Chandran and F. Grandoni, *Refined memorization for vertex cover.* Information Processing Letters, 93 (2005) 125-131.

[50] P. Charbit, S.ThomassÂt́e and A.Yeo, *The minimum feedback arc set problem is NP-hard for Tournaments.* To appear in Combinatorics, Probability and Computing.

[51] J. Chen, *Parameterized Computation and Complexity: A New Approach Dealing with NP-Hardness.* Journal of Computer Science and Technology 20(1): 18-37 (2005).

[52] J. Chen, F. V. Fomin, Y. Liu, S. Lu and Y. Villanger, *Improved Algorithms for the Feedback Vertex Set Problems.* To appear in the Proccedings of the WADS 2007.

[53] J. Chen, D. K. Friesen, W. Jia and I. A. Kanj, *Using Nondeterminism to Design Efficient Deterministic Algorithms.* In the in proceedings of 21st Foundations of Software Technology and Theoretical Computer Science (FSTTCS), Lecture Notes in Computer Science, **2245** (2001) 120-131.

[54] J. Chen. Y. Liu, S. Lu, B. O'Sullivan and I. Razgon, *A fixed-parameter algorithm for the directed feedback vertex set problem.* Journal of ACM 55(5), (2008).

[55] J.Chen, I. A. Kanj, *Improved Exact Algorithms for MAX-SAT.* In the proceedings of 5th Latin American Symposium (LATIN), Lecture Notes in Computer Science, **2286** (2002) 341-355.

[56] J. Chen and I. A. Kanj, *Constrained minimum vertex cover in bipartite graphs: complexity and parameterized algorithms.* Journal of Computer and System Sciences 67(4): 833-847 (2003).

[57] J. Chen, I. A. Kanj and W. Jia, *Vertex Cover, Further Observations and Further Improvements.* Journal of Algorithms **41** (2001) 280-301.

[58] J. Chen, I. A. Kanj, and W. Jia, *Vertex cover: further observations and further improvements.* Journal of Algorithms, 41 (2001), 280-301.

[59] J. Chen, I. A. Kanj and G. Xia, *Improved Parameterized Upper Bounds for Vertex Cover.* In the proceedings of 31st International Symposium on Mathematical Foundations of Computer Science (MFCS'06). LNCS 4162: 238-249 (2006).

[60] B. Chor, M. Fellows, D. W. Juedes. *Linear Kernels in Linear Time, or How to Save k Colors in $O(n^2)$ Steps.* In the proceedings of 30th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'04). LNCS 3353 : 257-269 (2004).

[61] N. Christofides, *An algorithm for the chromatic number of a graph.* Computer J., 14:38-39, 1971.

[62] F.R.K. Chung, *Separator theorems and their applications.* In Paths, flows, and VLSI-layout (Bonn, 1988), Series Algorithms Combin., 9 (1990), 17–34, Springer, Berlin.

[63] V. Contizer, *Computing Slater rankings using similarities among candidates.* Technical Report RC23748, IBM Thomas J Watson Research Centre, NY 2005.

[64] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, *Introduction to Algorithms*. MIT Press 2001.

[65] B. Courcelle, *The Monadic second-order logic of graphs I: recognizable sets of finite graphs*. Information and Computation 85 (1990), 12–75.

[66] B. Courcelle, *The monadic second-order logic of graphs III: tree-decompositions, minor and complexity issues*. Informatique Théorique et Applications (ITA) 26 (1992), 257–286.

[67] V. Dahllöf and P. Jonsson, *An algorithm for counting maximum weighted independent sets and its applications*. In 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2002), ACM and SIAM, 2002, pp. 292–298.

[68] V. Dahllöf, P. Jonsson, and M. Wahlström, *Counting models for 2SAT and 3SAT formulae*, Theoretical Computer Science, 332 (2005), pp. 265–291.

[69] F. Dehne, M. Fellows, M. Langston, F. Rosamond and K. Stevens, *An $O(2^{O(k)}n^3)$ FPT Algorithm for the Undirected Feedback Vertex Set Problem*. In the proceedings of 11th International Computing and Combinatorics Conference (COCOON), Lecture Notes in Computer Science **3595** (2005) 859-869.

[70] F. Dehne, M. Fellows, F. Rosamond, P. Shaw, *Greedy Localization, Iterative Compression, Modeled Crown Reductions: New FPT Techniques, an Improved Algorithm for Set Splitting, and a Novel $2k$ Kernelization for Vertex Cover*. In the proceedings of the First International Workshop on Parameterized and Exact Computation (IWPEC), Lecture Notes in Computer Science **3162** (2004) 271-280.

[71] G. Ding, T. Johnson, and P. Seymour, *Spanning trees with many leaves*. Journal of Graph Theory 37 (2001), 189–197.

[72] R. Downey. *Parameterized Complexity for the Skeptic*. In the Proceedings of of 18th IEEE Conference on Computational Complexity: 147-169 (2003).

[73] R. Downey, P. Evans and M. Fellows. Parameterized learning complexity. *Proc. Sixth ACM Workshop on Computational Learning Theory (COLT)*, pp. 51–57, ACM Press, 1993.

[74] R. G. Downey and M. R. Fellows, "Fixed Parameter Intractability," *Proceedings of the Seventh Structure in Complexity Theory conference*, (1992) 36-49.

[75] R. G. Downey and M. R. Fellows, "Parameterized Computational Feasibility", *Feasible Mathematics II, P. Clote and J. Remmel (eds.)* Birkhauser, Boston (1995) 219-244.

[76] R. G. Downey and M. R. Fellows, "Fixed Parameter Tractability and Completeness I: Basic Theory," *SIAM Journal of Computing* **24** (1995) 873–921.

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

[78] R. G. Downey and M. R. Fellows. *Threshold Dominating Sets and an improved characterization of W[2].* Theoretical Computer Science **209(1-2)** (1998) 123-140.

[79] R. Downey and M. Fellows. *Parameterized Complexity.* Springer-Verlag, (1999).

[80] R. Downey, M. Fellows, B. Kapron, M. Hallett and H.T. Wareham. The parameterized complexity of some problems in logic and linguistics. *Proceedings of the Symposium on the Logical Foundations of Computer Science*, Springer Verlag, Lecture Notes in Computer Science, vol. 813 (1994), 89–100.

[81] R. G. Downey, M. R. Fellows and V. Raman. *The complexity of irredundant sets parameterized by size.* Discrete Applied Mathematics **100(3)** (2000) 155-167.

[82] R. G. Downey, M. R. Fellows, A. Vardy and G. Whittle. *The parametrized complexity of some fundamental problems in coding theory.* SIAM Journal on Computing **29(2)** (1999) 545-570.

[83] N. Deo, M. S. Krishnamoorthy and M. A. Langston, "Exact and Approximate Solutions for the Gate Matrix Layout Problem", *IEEE Transactions on Computer-Aided Design* **6** (1987) 79-84.

[84] M. Drescher and A. Vetta, *An approximation algorithm for the maximum leaf spanning arborescence problem.* To appear in Transaction on Algorithms (2008).

[85] R. Duh and M. Fürer. *Approximation of k-set cover by semi-local optimization.* In the Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC), (1997) 256-264.

[86] C. Dwork, R. Kumar, M. Naor and D. Sivakumar, 'Rank Aggregation Revisited', Manuscript available at: *www.cs.northwestern.edu/ ∼ kao/cs395-ecommerce/reading_ document_ ranking/ecom.document_ ranking.dwork.pdf.*

[87] J. Edmonds. *Paths, trees and flowers.* Canadian Journal of Mathematics **17** (1965) 449-467.

[88] D. Eppstein. *3-coloring in time* $O(1.3289^n)$. Journal of Algorithms 54 (2): 168-204, (2005).

[89] D. Eppstein. *Small Maximal Independent Sets and Faster Exact Graph Coloring.* In the Proceedings of 7th Workshop on Algorithms and Data Structures (WADS). Lecture Notes in Computer Science 2125: 462-470 (2001).

[90] D. Eppstein. *Quasiconvex analysis of multivariate recurrence equations for backtracking algorithms.* ACM Transactions on Algorithms 2(4), 492-509 (2006).

[91] P. Erdös. *Problems and Results in Combinatroial Analysis and Graph Theory.* Discrete Mathematics 72: 81-92 (1988).

[92] P. Erdos and L. Posa, 'On the Maximal Number of Disjoint Circuits of a Graph', *Publ Math. Debrecen* **9** (1962) 3-12.

[93] P. Erdos and L. Posa, 'On Independent Circuits Contained in a Graph', *Canadian Journal of Mathematics* **17** (1965) 347-352.

[94] V. Estivill-Castro, M.R. Fellows, M.A. Langston, and F.A. Rosamond, FPT is P-Time Extremal Structure I. Proc. ACiD (2005), 1–41.

[95] G. Even, J. (Seffi) Naor, B. Schieber, M. Sudan, 'Approximating Minimum Feedback Sets and Multicuts in Directed Graphs', *Algorithmica* , **20** (1998) 151-174.

[96] R. J. Faudree, A. Gyárfas, R. H. Schelp and Z. Tuza. *Induced Matchings in Bipartite Graphs.* Discrete Mathematics 78: 83-87 (1989).

[97] U. Feige. *A threshold of ln n for approximating set cover.* Journal of the ACM **45(4)** (1998) 634-652.

[98] M. Farber, H. Hahn, P. Hell and D. Miller, "Concerning the Achromatic Number of Graphs", Journal of Combinatorial Theory, Series B **40** 21-39 (1986).

[99] M. R. Fellows, private communication.

[100] M. Fellows, Private communications, 2005-2006.

[101] M. R. Fellows, "The Robertson-Seymour theorems: a survey of applications", in *Contemporary Mathematics* Vol **89**, AMS (1989) 1-18.

[102] M. Fellows, M. Hallett, C. Korostensky, U. Stege, 'Analogs and Duals of the MAST Problem for Sequences and Trees', *Journal of Algorithms*, **49** (1) (2003) 192-216.

[103] M. R. Fellows, M. T. Hallett and H. T. Wareham, "DNA physical mapping: three ways difficult", *In Proceedings of the European Symposium on Algorithms*, Lecture Notes in Computer Science, Springer Verlag **726** (1993) 157-168.

[104] M. Fellows, D. Hermelin and F. Rosamond. *On the fixed-parameter intractability and tractability of multiple-interval graph properties.* Manuscript, 2007.

[105] M. R. Fellows and M. A. Langston, "Nonconstructive Tools for Proving Polynomial-Time Decidability", *Journal of the Association of Computing Machinery* **35** (1988) 727-739.

[106] M. R. Fellows and M. A. Langston, "On Search, Decision, and the Efficiency of Polynomial-Time Algorithms", *Journal of Computer and System Sciences* **49** (1994) 769-779.

[107] M. R. Fellows and M. A. Langston, "An Analogue of the Myhill-Nerode Theorem and its Use in Computing Finite Basis Characterizations", *In Proceedings of the Symposium on Foundations of Computer Science, FOCS* (1989) 520-525.

[108] M.R. Fellows and M.A. Langston, On well-partial-order theory and its applications to combinatorial problems of VLSI design. SIAM Journal on Discrete Mathematics 5 (1992), 117–126.

[109] M.R. Fellows, C. McCartin, F.A. Rosamond, and U. Stege, Coordinated kernels and catalytic reductions: An improved FPT algorithm for max leaf spanning tree and other problems. Lect. Notes Comput. Sci. 1974 (2000), 240–251.

[110] H. FERNAU. *Parameterized Algorithmics for d-*HITTING SET. Manuscript.

[111] H. FERNAU, *Parameterized algorithms: A graph-theoretic approach*, Apr. 2005. Habilitationsschrift, UniversitÂĺat TÂĺubingen, TÂĺubingen, Germany.

[112] H. FERNAU, *Edge dominating set: efficient enumeration-based exact algorithms*, in Proceedings of the 2nd International Workshop on Parameterized and Exact Computation (IWPEC 2006), vol. 4169 of LNCS, Springer, Berlin, 2006, pp. 142–153.

[113] H. FERNAU AND R. NIEDERMEIER. *An Efficient Exact Algorithm for Constraint Bipartite Vertex Cover.* Journal of Algorithms 38(2): 374-410 (2001).

[114] J. Flum and M. Grohe. *Parameterized Complexity Theory.* Springer-Verlag, (2006).

[115] F. V. Fomin, S. Gaspers and A. V. Pyatkin. *Finding a Minimum Feedback Vertex Set in time $O(1.7548^n)$*. In the Proceeding of 2nd International Workshop on Parameterized and Exact Computation (IWPEC), LNCS **4169** (2006) 184-191.

[116] F. V. Fomin, F. Grandoni, and D. Kratsch. Measure and Conquer: A Simple $O(2^{0.288n})$ Independent Set Algorithm. In the proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA): 18-25, (2006).

[117] F. V. Fomin, F. Grandoni, and D. Kratsch. *Some new techniques in design and analysis of exact (exponential) algorithms.* Bulletin of the EATCS 87: 47-77 (2005).

[118] F. V. Fomin, F. Grandoni and D. Kratsch, *Measure and Conquer: Domination – A Case Study*, in Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP 2005), vol. 3580 of LNCS, Springer, Berlin, 2005, pp. 191–203.

[119] F. V. Fomin, S. Gaspers, S. Saurabh, *Branching and Treewidth Based Exact Algorithms*,in Proceedings of the 17th International Symposium on Algorithms and Computation (ISAAC 2006), vol. 4288 of LNCS, Springer, Berlin, 2006, pp. 16–25.

[120] F. V. Fomin, D. Kratsch and G. J. Woeginger, *Exact (exponential) algorithms for the dominating set problem*, in Proceedings of the 30th Workshop on Graph Theoretic Concepts in Computer Science (WG 2004), vol. 3353 of LNCS, Springer, Berlin, 2005, pp. 245–256.

[121] F. V. Fomin and K. Høie, *Pathwidth of cubic graphs and exact algorithms*, Information Processing Letters, 97 (2006), pp. 191–196.

[122] F. V. Fomin and D. M. Thilikos, 'Dominating Sets in Planar Graphs: Branch-Width and Exponential Speed-up', in *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, (2003) 168-177.

[123] M. Fürer and S. P. Kasiviswanathan, *Algorithms for counting 2-SAT solutions and colorings with applications*, in Electronic Colloquium on Computational Complexity (ECCC), vol. 33, 2005.

[124] F. Gavril and M. Yannakakis. *Edge Dominating Sets in Graphs.* SIAM Journal on Applied Mathematics 38(3): 364-372 (1980).

[125] G. Galbiati, F. Maffioli, and A. Morzenti, A short note on the approximability of the maximum leaves spanning tree problem. Information Processing Letters 52 (1994), 45–49.

[126] G. Galbiati, A. Morzenti, and F. Maffioli, On the approximability of some maximum spanning tree problems. Theoretical Computer Science 181 (1997), 107–118.

[127] M. R. Garey and D. S. Johnson. *Computer and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, San Francisco, CA., (1979).

[128] F. Gavril and M. Yannakakis, *Edge dominating sets in graphs*, SIAM Journal on Applied Mathematics, 38 (1980), pp. 364–372.

[129] J. P. Georges, M. D.Halsey, A. M. Sanaulla, M. A. Whittlesey.*Edge Domination and Graph Structure.* Cong. Numer. 76: 127-144 (1990).

[130] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs.* Academic Press, New york, 1980.

[131] R. L. Graham and J. H. Spencer,"A Constructive solution to a tournament problem", *Canadian Mathematical Bulletin* **14** (1971) 45-48.

[132] J. Gramm, J. Guo, F. Hüffner and R. Niedermeier. *Automated Generation of Search Tree Algorithms for Hard Graph Modification Problems.* Algorithmica 39(4): 321-347 (2004).

[133] J. Gramm, E. A. Hirsch, R. Niedermeier and P. Rossmanith. *Worst-case upper bounds for MAX-2-SAT with an application to MAX-CUT.* Discrete Applied Mathematics 130 (2): 139-155 (2003).

[134] F. Grandoni, *A note on the complexity of minimum dominating set*, Journal of Discrete Algorithms, 4 (2006), pp. 209–214.

[135] J.R. Griggs and M. Wu, Spanning trees in graphs of minimum degree four or five. Discrete Mathematics 104 (1992), 167–183.

[136] E. Ya. Grinberg and Ya. Ya. Dambit, 'Nekotorye Svoistva Grafov Soderzhashchikh Kontury' [Russian: Some Properties of Directed Graphs With Circuits], *Latviiskii Mathematicheskii Ezhegodnile*, **2** (1966) 65-70.

[137] D. L. GRINSTEAD, P. J. SLATER, N. A. SHERWANI, N. D. HOLMES. *Efficient Edge Domination Problems in Graphs.* Information Processing Letters 48(5): 221-228 (1993).

[138] J. Guo, J. Gramm, F. Huffner, R. Niedermeier, and S. Wernicke, 'Improved Fixed-Parameter Algorithms for Two Feedback Set Problems', in *Proceedings of 9th Workshop on Algorithms and Data Structures (WADS)*, Lecture Notes in Computer Science **3608** (2005) 158-168.

[139] J. Guo, R. Niedermeier and S. Wernicke. *Parameterized complexity of generalized Vertex Cover problems.* In the Proceeding of 9th International Workshop Algorithms and Data Structures (WADS), LNCS **3608** (2005) 36-48.

[140] Y. GUREVICH AND S. SHELAH. *Expected Computation Time for Hamiltonian Path Problem.* SIAM Journal on Computing 16(3): 486-502, (1987).

[141] G. Gutin, The radii of n-partite tournaments. Math. Notes 40 (1986), 743–744.

[142] G. Gutin and A. Yeo, Some Parameterized Problems on Digraphs. To appear in The Computer Journal.

[143] T. Gallai, 'On Directed Paths and Circuits', *in Theory of Graphs, Proceedings Colloquium Tihnay, P Erdős, G. Katona Eds.*, (1968) 115-118.

[144] F. Harary. *Graph Theory.* Addison-Wesley Publishing Company (1969).

[145] M. HELD AND R. M. KARP. *A dynamic programming approach to sequencing problems.* Journal of SIAM 10, 196Âŋ210 (1962).

[146] E. HORWITZ AND S. SAHNI. *Computing partitions with applications to the Knapsack problem.* Journal of the ACM 21, 277-292 (1974).

[147] M. HUJTER AND Z. TUZA. *The number of Maximal Independent Sets in Triangle-Free Graphs.* SIAM Journal on Discrete Mathematics 6(2): 284-288 (1993).

[148] O. H. Ibarra and C. E. Kim, “Fast Approximation Algorithms for the Knapsack and sum of subset problems”, *Journal of the ACM* **22** (1975) 463-468.

[149] A. Itai and M. Rodeh, ‘Finding a Minimum Circuit in a Graph’, *SIAM Journal on Computing*, **7** (1978) 413-423.

[150] K. Iwama and S. Tamaki. *Improved upper bounds for 3-SAT.* In the proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA): 328- , (2004).

[151] K. Iwama, *Worst-case upper bounds for k-SAT*, Bulletin of the EATCS, 82 (2004), pp. 61–71.

[152] T. Jian, *An* $O(2^{0.304n})$ *algorithm for solving maximum independent set problem*, IEEE Transactions on Computers, 35 (1986), pp. 847–851.

[153] David S. Johnson. *Approximation Algorithms for Combinatorial Problems.* Journal of Computer and System Sciences **9(3)** (1974) 256-278.

[154] D. S. Johnson, M. Yannakakis and C. H. Papadimitriou. *On Generating all Maximal Independent Sets.* Information Processing Letters 27: 119-123 (1988).

[155] S. Jukna. *Extremal Combinatorics.* Springer-Verlag (2001).

[156] I. Kanj, M. Pelsmajer and M. Schaefer, “Parameterized Algorithms for Feedback Vertex Set", To appear in the *Proceedings of IWPEC-2004*.

[157] I. Kanj and L. Perkovic, ‘Improved Parameterized Algorithms for Planar Dominating Set’, in *Mathematical Foundations of Computer Science (MFCS)*, Lecture Notes in Computer Science **2420** (2002) 399-410.

[158] D. Karger, R. Motwani and G. D. S. Ramkumar, “On approximating the longest path in a graph”, *In Proceedings of the Workshop on Algorithms and Data Structures* (Montreal, Quebec), Lecture Notes in Computer Science **709** Springer-Verlag (1993) 421-432.

[159] R. M. Karp, ‘Reducibility Among Combinatorial Problems’, *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher Eds., New York Plenum Press, (1972) 85-103.

[160] S. Khot and V. Raman. *Parameterized complexity of finding Subgraphs with hereditary properties.* Theoretical Computer Science **289(2)** (2002) 997-1008.

[161] N. G. Kinnersley, The vertex separation number of a graph equals its path-width, Information Processing Letters 42 (1992), 345–350.

[162] L. M. Kirousis and C. H. Papadimitriou, Interval graphs and searching, Discrete Mathematics 55 (1985), 181–184.

[163] D.J. Kleitman and D.B. West, Spanning trees with many leaves. SIAM Journal on Discrete Mathematics 4 (1991), 99–106.

[164] T. Kloks, C. M. Lee and J. Liu, 'Feedback Vertex Sets and Disjoint Cycles in Planar (di) Graphs', *Optimization Online*, Mathematical Programming Society (2001).

[165] J. Kneis, D. Mölle, S. Richter and P. Rossmanith . *Algorithms Based on the Treewidth of Sparse Graphs.* In the proceedings of Workshop on Graph-Theoretic Concepts in Computer Science (WG). LNCS 3787: 385-396, (2005).

[166] J. Kneis, A. Langer and P. Rossmanith, *A new algorithm for finding trees with many leaves.* To appear in Proceedings of ISAAC 2008, LNCS.

[167] P. G. Kolaitis and M. N. Thakur, "Approximation properties of NP minimization classes", *Journal of Computer and System Sciences* **50** (1995) 391-411.

[168] G. Kortsarz and D. Peleg. *On Choosing a Dense Subgraph.* In the Proceeding of 34th Annual Symposium on Foundations of Computer Science (FOCS), (1993) 692-701.

[169] A. V. Kostochka, M. J. Pelsmajer and D. B. West. *A list analogue of equitable coloring.* Journal of Graph Theory 44, 166-177 (2003).

[170] E. L. Lawler. *A Note on the Complexity of the Chromatic Number.* Information Processing Letters 5 (3): 66-67 (1976).

[171] N. Linial and D. Sturtevant (1987). Unpublished result.

[172] D. Lokshtanov and C. Sloper. *Fixed Parameter Set Splitting, Linear Kernel and Improved Running Time.* In Proceedings of Algorithms and Complexity in Durham (ACID), 105-113, (2005).

[173] L. Lovàsz. *On the ratio of optimal fractional and integral covers.* Discrete Mathematics **13** (1975) 383-390.

[174] H.-I. Lu and R. Ravi, Approximating maximum leaf spanning trees in almost linear time. Journal of Algorithms 29 (1998), 132–141.

[175] E. M. Luks. *Isomorphism of Graphs of Bounded Valence can be Tested in Polynomial Time.* Journal of Computer System Sciences 25(1): 42-65 (1982).

[176] D. Marx, 'Chordal Deletion is Fixed Parameter Tractable', manuscript.

[177] M. Mahajan and V. Raman. *Parameterizing above Guaranteed Values: MaxSat and MaxCut.* Journal of Algorithms 31(2): 335-354, (1999)

[178] M. Mahajan, V. Raman and S. Sikdar, 'Parameterizing MAX SNP Problems Above Guaranteed Values', *in the Proceedings of 2nd International Workshop on Parameterized and Exact Computation (IWPEC 2006)*, Lecture Notes in Computer Science, **4169** (2006) 38-49 Springer-Verlag.

[179] N. Meggido, "Linear Programming in Linear Time when the Dimension is Fixed", *Journal of the ACM* **31** (1984) 114-127.

[180] N. Meggido, S. L. Hakimi, M. R. Garey, D. S. Johnson and C. H. Papadimitriou, "The complexity of searching a graph ", *Journal of the ACM* **35** (1988) 18-44.

[181] N. Meggido and U. Vishkin, "On Finding a Minimum Dominating Set in a Tournament", *Theoretical Computer Science* **61**, (1988) 307–316.

[182] W. Meyer.*Equitable coloring.* American Mathematical Monthly 80, 920-922 (1973).

[183] R. H. Möhring, Graph problems related to gate matrix layout and PLA folding. In Computational Graph Theory, vol. 7 of Comput. Suppl., Springer, Vienna, (1990), 17–51.

[184] D. Mölle, S. Richter and P. Rossmanith. *A Faster Algorithm for the Steiner Tree Problem.* In the Proceedings of 23rd Symposium on Theoretical Aspects of Computer Science. Lecture Notes in Computer Science 3884: 561-570 (2006).

[185] J. W. Moon and L. Moser. *On Cliques in Graphs.* Israel Journal of Mathematics 3: 23-28 (1965).

[186] J. W. Moon. *On maximal Transitive Subtournaments.* Proceedings of the Edinburgh Mathematical Society 17: 345-349 (1971).

[187] B. Monien, "How to find long paths efficiently?", *Annals of Discrete Mathematics*, **25** (1985) 239-254.

[188] B. Monien and R. Schulz, 'Four Approximation Algorithms for the Feedback Vertex Set Problem', in *Proceedings of the 7th Conference on Graphtheoretic Concepts in Computer Science*, Hanser Verlag (1981) 315-326.

[189] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms.* Oxford Lecture Series in Mathematics and Its Applications, Oxford University Press, (2006).

[190] R. Niedermeier and P. Rossmanith, 'An efficient Fixed Parameter Algorithm for 3-Hitting Set', *Journal of Discrete Algorithms*, **1** (1) (2003) 89-102.

[191] R. Niedermeier and P. Rossmanith, *Upper bounds for vertex cover further improved*, in Proceedings of the 16th International Symposium on Theoretical Aspects of Computer Science (STACS 1999), vol. 1563 of LNCS, Springer, Berlin, 1999, pp. 561–570.

[192] ——, *On efficient fixed-parameter algorithms for weighted vertex cover*, Journal of Algorithms, 47 (2003), pp. 63–77.

[193] C. H. Papadimitriou and M. Yannakakis, "On Limited Nondeterminism and the Complexity of the V-C dimension", *Proceedings of Eighth Structure in Complexity Theory conference*, (1993) 12-18; to appear in JCSS.

[194] V. Petrovic and C. Thomassen, Kings in k-partite tournaments. Discrete Mathematics 98 (1991), 237–238.

[195] S. Poljak and D. Turzik, ‘A Polynomial Algorithm for Constructing a Large Bipartite Subgraph, with an Application to a Satisfiability Problem’,*Canad. J. Math*, **34** (3) (1982) 519-524.

[196] E. Prieto. *The Method of Extremal Structure on the k-Maximum Cut Problem.* In the proceedings of Computing: The Australasian Theory Symposium (CATS): 119-126, (2005).

[197] V. Raman, “Some Hard Problems in (Weighted) Tournaments”, *Proceedings of the Fifth National Seminar on Theoretical Computer Science*, Bombay, India (1995) 115-122.

[198] V. Raman, ‘Parameterized Complexity,’ in *Proceedings of the 7th National Seminar on Theoretical Computer Science*, (1997) 1-18.

[199] V. Raman, S. Saurabh and S. Sikdar *Efficient Exact Algorithms through Enumerating Maximal Independent Sets and Other Techniques.* To appear in Theory of Computing Systems. (2006).

[200] B. Randerath and I. Schiermeyer. *Exact Algorithms for Minimum Dominating Set.* Technical Report, zaik-469, Zentrum für Angewandte Informatik Köln, (2004).

[201] I. Razgon. *Exact Computation of Maximum Induced Forest.* To appear in the Proceedings of 10th Scandinavian Workshop on Algorithm Theory (SWAT). Lecture Notes in Computer Science (2006).

[202] B. Reed, K. Smith and A. Vetta. *Finding Odd Cycle Transversals.* Operations Research Letters 32: 299-301 (2004).

[203] N. Robertson and P. D. Seymour, Graph minors-a survey. In I. Anderson (Ed.) *Surveys in Combinatorics*, Cambridge Univ. Press, (1985), 153–171.

[204] N. Robertson and P. D. Seymour, Graph minors I: Excluding a forest. Journal of Combinatorial Theory Series B 35 (1983), 39–61.

[205] N. Robertson and P. D. Seymour, “Graph Minors II. Algorithmic Aspects of tree-width”, *Journal of Algorithms*, **7** (1986) 309-322.

[206] N. Robertson and P. D. Seymour, "Graph Minors XIII. The Disjoint Paths Problem," *Journal of Combinatorial Theory Series B* **63** (1995) 65-110.

[207] N. Robertson and P. D. Seymour, "Graph Minors XV. Wagner's Conjecture", **********

[208] J. M. Robson. *Algorithms for Maximum Independent Set.* Journal of Algorithms 7: 425 - 440 (1986).

[209] J. M. Robson. *Finding a Maximum Independent Set in time $O^*(2^{n/4})$?* Technical Report 1251-01, LaBRI, Université Bordeaux I, (2001).

[210] K. Rosen, Handbook of Discrete and Combinatorial Mathematics, CRC Press, 2000.

[211] S. Sahni, "Algorithms for scheduling independent tasks," *Journal of the ACM* **23** (1976) 116-127.

[212] U. Schöning, *Algorithmics in exponential time*, in Proceedings of the 22nd International Symposium on Theoretical Aspects of Computer Science (STACS 2005), vol. 3404 of LNCS, Springer, Berlin, 2005, pp. 36–43.

[213] R. Solis-Oba, 2-approximation algorithm for finding a spanning tree with the maximum number of leaves. Lect. Notes Comput. Sci. 1461 (1998), 441–452.

[214] E. Speckenmeyer, 'On Feedback Problems in Digraphs', *in Proceedings of the 15th International Workshop WG'89*, Lecture Notes in Computer Science, **411** (1989) 218-231 Springer-Verlag.

[215] A. Steger and M. Yu. *On Induced Matchings.* Discrete Mathematics 120: 291-295 (1993).

[216] L. J. Stockmeyer and V. V. Vazirani. *NP-Completeness of Some Generalizations of the Maximum Matching Problem.* Information Processing Letters 15(1): 14-19 (1982).

[217] R. E. Tarjan and A. E. Trojanowski, *Finding a maximum independent set*, SIAM Journal on Computing, 6 (1977), pp. 537–546.

[218] J. van Leeuwen, Graph Algorithms, in “Handbook of Theoretical Computer Science. Vol A, Algorithms and Complexity Theory” North Holland, Amsterdam (1990) 527-631.

[219] V. V. Vazirani. *Approximation Algorithms.* Springer-Verlag (2001).

[220] H. J. Voss, ‘Some Properties of Graphs Containing k Independent Circuits‘, in *Proceedings Colloq. Tihany*, Academic Press (1968) 321-334.

[221] H. J. Voss and H. Walter, ‘Ober Kreise in Graphen, VEB Deutscher Verlag der Wissenschaften’, Berlin 1974, Teil II, Kapitel III.

[222] M. Wahlström. *Exact algorithms for finding minimum transversals in rank-3 hypergraphs.* Journal of Algorithms 51(2): 107 - 121 (2004).

[223] D. B. West. *Introduction to Graph Theory.* Prentice Hall, Second Edition (2001).

[224] A. Wigderson. *P, NP and Mathematics - a computational complexity perspective.* In the Proceedings of the International Congress of Mathematicians (ICM 06). Volume I, EMS Publishing House, Zurich, 665-712 (2007).

[225] R. Williams, *A new algorithm for optimal 2-constraint satisfaction and its implications*, Theoretical Computer Science, 348 (2005), pp. 357–365.

[226] G. Woeginger. *Exact algorithms for NP-hard problems: A survey.* In *Combinatorial Optimization—Eureka! You shrink!.* Lecture Notes in Computer Science 2570: 185-207 (2003).

[227] G. Woeginger. *Space and Time Complexity of Exact Algorithms: Some Open Problems.* In the Proceedings of the 1st International Workshop on Exact and Parameterized Algorithms (IWPEC). Lecture Notes in Computer Science 3162: 281-290 (2004).

[228] B. Y. Wu and K. Chao, *Spanning Trees and Optimization Problems*, CRC Press, 2003.

[229] M. Yannakakis, ‘Node and Edge-Deletion NP-Complete Problems’, in *Proceedings of the 10th Annual ACM Symposium on Theory of Computing (STOC)*, (1978) 253-264.