PARAMETERIZED ALGORITHMS ON DIGRAPH AND CONSTRAINT SATISFACTION PROBLEMS

A thesis submitted to Royal Holloway, University of London
for the degree of Doctor of Philosophy
in the Faculty of Science

May 2010

By
EUN JUNG KIM
Department of Computer Science

Declaration

I Eun Jung Kim hereby certify that this thesis is the record of work carried out by me and that it has not been submitted to any previous application for a higer degree. Wherever contributions of others are involved, every effort is made to indicate this clearly, with due reference to the literature and acknowledgement of collaborative research and discussions.

Abstract

While polynomial-time approximation algorithms remain a dominant notion in tackling computationally hard problems, the framework of parameterized complexity has been emerging rapidly in recent years. Roughly speaking, the analytic framework of parameterized complexity attempts to grasp the difference between problems which admit $O(c^k \cdot poly(n))$-time algorithms such as Vertex Cover, and problems like Dominating Set for which essentially brute-force $O(n^k)$-algorithms are best possible until now. Problems of the former type is said to be fixed-parameter tractable (FPT) and those of the latter type are regarded intractable. In this thesis, we investigate some problems on directed graphs and a number of constraint satisfaction problems (CSPs) from the parameterized perspective.

We develop fixed-parameter algorithms for some digraph problems. In particular, we focus on the basic problem of finding a tree with certain property embedded in a given digraph. New or improved fpt-algorthms are presented for finding an out-branching with many or few leaves (Directed Maximum Leaf, Directed Minimum Leaf problems). For acyclic digraphs, Directed Maximum Leaf is shown to allow a kernel with linear number of vertices. We suggest a kernel for Directed Minimum Leaf with quadratic number of vertices. An improved fpt-algorithm for finding k-Out-Tree is presented and this algorithm is incorporated as a subroutine to obtain a better algorithm for Directed Minimum Leaf.

In the second part of this thesis, we concentrate on several CSPs in which we want to maximize the number of satisfied constraints and consider parameterization “above tight lower bound” for these problems. To deal with this type of parameterization, we present a new method called SABEM using probabilistic approach and applying harmonic analysis on pseudo-boolean functions. Using SABEM we show that a number of CSPs admit polynomial kernels, thus being fixed-parameter tractable. Moreover, we suggest some problem-specific combinatorial approaches to Max-2-Sat and a wide special class of Max-Lin2, which lead to a kernel of smaller size than what can be obtained using SABEM for respective problems.

Acknowledgements

It is a pleasure to record my debts to those with whom I have been fortunate enough to be around. My first and deepest gratitude goes out to my supervisor, Gregory Gutin. After years as a PhD student, all the more I realize how lucky I am to have him as a supervisor. Gregory led me to the field of research in many ways and his drive and sincerity on research has become my discipline. Support and encouragement were furnished so timely by him, for which I am grateful. I also wish to acknowledge my appreciation to my thesis committee, Stephan Kreutzer and Costas S. Iliopoulos, for insightful comments and discussion on this work.

To properly thank everyone to whom I am indebted during the academic journey would involve adding a whole new chapter to the thesis. However, I don't want to miss the opportunity to recognize the creative and patient efforts of my coauthors; Noga Alon, Robert Crowston, Jean Daligault, Peter Dunkelmann, Nathann Cohen, Fedor Fomin, Arvind Gupta, Mark Jones, Mehdi Karimi, Michael Lampis, Valia Mitsou, Matthias Mnich, Sebastian Ordyniak, Arash Rafiey, Igor Razgon, Imre Ruzsa, Saket Saurabh, Arezou Soleimanfallah, Stefan Szeider, Ryan Williams and Anders Yeo. I enjoyed the work and learned so much from them. Particular thanks to Anders Yeo for countless invaluable discussions, which helped me learn things and how to think. I cannot thank enough Kyung Chul Chae; his grateful kindness and support made it possible to start my PhD.

The geographic distribution of my friends is another evidence that we are living a globalized era. You guys made my life easier and brighter. Special thanks to all of you. A very special thanks to one person in particular.

Every time passing tells me how much I owe to my parents. Keep your life and work just as you have done, and I cannot imagine better encouragement. (I want to join your Himalayan tracking, but am afraid to drag your steps.) As always, my brother has been there and I wish you the best of luck. I congratulate my sister, my best friend and the best mentor, for her new track just beginning.

A final word goes to the piano which walked into my room one day and lit up another dimension of my life.

Contents

Chapter 1

Introduction

1.1 Parameterized Framework

Computationally intractable problems are prevalent in applications. Especially when we look into a real-world problem which needs to be solved, it is quite likely that the problem does not allow a polynomial-time algorithm. Several approaches have been taken to deal with the situation. Quite so often the strategy is to sacrifice the quality of a solution for efficiency. Approximation algorithms always produce a solution within a guaranteed gap whereas typical randomized algorithms with error probability produce an optimal solution often enough, though not always.

Since its emergence in the early 1990s, the parameterized framework has proved itself as a fresh yet solid ground for tackling the apparently intractable problems and viewing their asymptotic behavior. The conscious form of parameterized complexity appeared with a series of papers by Downey and Fellows [49], but there are a number of early examples for fixed-parameter algorithms. A classic example is the $O^*(3^k)$[1] dynamic programming algorithm for the Steiner Tree Problem [50], where k is the number of terminal vertices.

The parameterized framework can be construed as a way of looking into the structure of an instance with a bird's eye view, instead of an ant's eye view on one-dimensional space. The classic complexity theory gives a dichotomy of P versus NP based on analysis of resource requirements in terms of the instance size only. The parameterized complexity captures another dimension of an instance, called a *p*arameter, and provides a framework to explain to what extent this additional dimension affects the resource requirement. If we are able to identify a parameter k which plays a crucial role in the seemingly unavoidable combinatorial explosion and hopefully the parameter can be assumed to have small

[1] we often use the notation $O^*(f(k))$ instead of $f(k)(kn)^{O(1)}$, i.e., O^* hides not only constants, but also polynomial coefficients.

values, we may solve the problem efficiently. In other words, the analytic framework of parameterized complexity attempts to grasp the difference between problems which admit $O(c^k \cdot poly(n))$-time algorithms such as Vertex Cover, and problems like Dominating Set for which essentially brute-force $O(n^k)$-algorithms are best possible until now. Problems of the former type is said to be fixed-parameter tractable (FPT) and the latter type is regarded intractable in parameterized sense.

The focus of our work will be directed at designing fixed-parameter algorithms for hard combinatorial problems. Plenty of algorithmic methods have been developed for parameterized problems; kernelization, depth-bounded search tree, iterative compression, color-coding and dynamic programming for example. In particular, recent years have witnessed a rapid progress in the study of kernelization. A good part of our work will be devoted for kernelization issue and we attempt to provide a new method for kernelization in Part II. An excellent overview of much recent work on kernelization can be found in Guo and Niedermeier [65].

1.2 Parameterized Algorithms for Digraph Problems

Due to its asymmetric nature, problems on directed graphs are frequently more difficult to settle than their counterparts on undirected graphs. In case we manage to design an algorithm for some digraph problem to match the efficiency of their undirected counterpart, it may be require a quite different approach or at least an additional idea. With this mind, we are not so surprised that the study of parameterized algorithms on digraphs problems tends to lag behind its undirected counterpart. Lagging behind, however, means an opportunity to catch up.

As problems on digraphs will be the focal point in the first part of the thesis, we provide a brief overview of the recent development in related research.

The Maximum Leaf problem is to find a spanning tree with the maximum number of leaves in a given undirected graph. The problem is well studied from both algorithmic [57, 60, 89, 104] and graph-theoretical [47, 81, 88] points of view. Note that Maximum Leaf in undirected setting is equivalent to the problem Connected Dominating Set. This problem finds a primary application in wireless ad hoc network [20], where we want to decide a set of points for routing. A small connected dominating set meets the requirementfor such relaying points and thus, can provide a backbone for communication flow. This problem has been studied from the parameterized complexity perspective as well and several authors [27, 51, 53] have designed fixed parameter tractable (FPT) algorithms for solving the parameterized version of Maximum Leaf (the k-Leaf problem): given a graph G and an integral parameter k, decide whether G has a spanning tree with at least k leaves.

The study of Directed k-Leaf has begun quite recently. Alon et al. [3, 2] proved that the problem is FPT for a wide family of digraphs including classes of strongly connected and acyclic digraphs. Bonsma and Dorn extended this result to all digraphs in [28], and improved the running time of the algorithm in [2] to $O^*(2^{k \log k})$ in [29]. Later on, Kneis et al. [82] proposed a simple yet elegant algorithm of running time $O^*(4^k)$, a big improvement for both directed and undirected k-Leaf problem. We further develop their algorithm and obtain an fpt-algorithm running in time $O^*(3.72^k)$, which shall be described in Chapter 3. The latest update on the running time race is $O^*(3.4575^k)$-algorithm for undirected graphs by Raible and Fernau [103].

Turning to the kernelization side, a kernel of size $3.75k$ is known [51] for undirected graphs. When we stretch for directed graphs, the landscape is quite different. Fernau et al. [54] proved that no polynomial kernel for Directed k-Leaf is possible unless the polynomial hierarchy collapses to the third level (they applied a recent breakthrough result of Bodlaender et al. [23]). Interestingly, if we specify the root, then the problem Rooted Directed k-Leaf admits a polynomial size kernel and Fernau et al. [54] obtained one of size $O(k^3)$. This was later improved to a quadratic one by Daligault and Thomassé [42]. The authors of [42] also presented an 92-approximation algorithm for Directed Maximum Leaf. To the best of our knowledge, no linear vertex-kernel has been proposed so far.

The k-Out-Tree problem is the problem of deciding, for a given parameter k and a given out-tree T on k vertices, whether an input digraph contains T as a subgraph. In their seminal work on Color Coding Alon, Yuster, and Zwick [8] provided fixed-parameter tractable (FPT) randomized and deterministic algorithms for k-Out-Tree. While Alon, Yuster, and Zwick [8] only stated that their algorithms are of runtime $O^*(2^{O(k)})$, it is easy to see (see Subsection 4.1) that their randomized and deterministic algorithms are of complexity $O^*((4e)^k)$ and $O^*(c^k)$, where $c \geq 4e$. In fact, the derandomization of Color Coding requires a huge blow-up in the constant.

The main results of [8], however, were a new algorithmic approach called Color Coding and a randomized $O^*((2e)^k)$ algorithm for deciding whether a digraph contains a path with k vertices (the k-Path problem). Chen et al. [31] proposed another approach, a randomized Divide-and-Conquer technique. The new approach allowed them to design a randomized $O^*(4^k)$-time algorithm for k-Path. The Divide-and-Conquer technique of Chen et al. [31] uses two colors. The colors are 'symmetric', i.e., both colors play similar role and the probability of coloring each vertex in one of the colors is 0.5. In Chapter 4, we further develop the technique of [31] by making it asymmetric, i.e., the two colors play different roles and the probability of coloring each vertex in one of the colors depends on the color. As a result, we refine the result of Alon, Yuster, and Zwick by obtaining randomized and deterministic algorithms for k-Out-Tree, of runtime $O^*(5.704^k)$

and $O^*(6.14^k)$, respectively.

Recent breakthrough results due to Koutis [83] and Williams [107] are based on an algebraic formulation of the k-Path problem. Koutis [83] reformulated k-Path as the problem of detecting square-free term in degree-k polynomial and obtained a randomized $O^*(2^{3k/2})$-time algorithm. Williams [107] extended his ideas culminating in a randomized $O^*(2^k)$-time algorithm. While the randomized algorithms based on Color Coding and Divide-and-Conquer are not difficult to derandomize, it is not the case for the algorithms of Koutis [83] and Williams [107]. Thus, it is unknown whether there are deterministic algorithms for k-Path of runtime $O^*(2^k)$ or even $O^*(2^{3k/2})$. In [84], Koutis and Williams suggested a new application of their method for k-Tree, achieving the running time of $O^*(2^k)$. The derandomization issue is yet to be resolved in this case as well.

The Directed Minimum Leaf problem is to find an out-branching with the minimum number of leaves in a given digraph. The Hamilton path problem is its special case and thus, Directed Minimum Leaf is NP-hard. Chapter 5 is devoted to the study of this problem.

Directed Minimum Leaf is of particular interest in computer database systems [45]. When a huge amount of information is stored in indexed tables, a naive way of retrieving information that satisfy a query would be to inspect every row for those columns specified by the query. To avoid such hassle, many database systems are equipped with indexes, which are conceptually similar to an index at the end of a book. For a frequently queried combination of columns, we may keep a list of information (i.e. combined values in the corresponding columns) and their locations. While providing indexes seems to be an all-round solution, the update of indexes to keep track of changes in the tables is another costly job and we prefer fewer indexes to update. Here the following problem arise: how can we decide the optimal set of indexes which also covers all frequently queried combinations of columns? This questions can be easily formulated as the problem of finding an out-branching with minimum number of leaves in an acyclic digraph. Demers and Downing [45] suggested a heuristic approach to this problem, but no argument or assertion has been made to provide the validity of their approach and to investigate its running time.

We give a simple proof in Section 5.1 that the problem Directed Minimum Leaf can be solved in polynomial time when the input graph is restricted to be acyclic. Then we examine a broader class of near-acyclic digraphs with respect to known digraph width measures and explore how far we can extend the polynomiality result. The hardness of Directed Minimum Leaf obtained in our work fits well with the implication of the recent papers [85, 87]; that only a relatively few NP-hard optimization problems on digraphs become tractable when restricted to digraphs of bounded directed width parameters. This is a sharp contrast to the situation with undirected graphs in which a vast body of NP-hard optimization problems becomes tractable on graphs of bounded tree-width. These negative outcome highlights again the difficulty in coping with digraphs and indicates a long way to go in order to match the success of tree-width for undirected graphs.

In the subsequent sections of Chapter 5, we will study the following parameterized version of Directed Minimum Leaf: given a digraph D and a parameter k, decide whether D has an out-branching with at least k internal vertices. This problem, denoted Directed k-Internal, was studied for undirected graphs by Prieto and Sloper [101, 102]. We demonstrate an algorithm of runtime $O^*(2^{O(k \log k)})$ for Directed k-Internal and another algorithm with running time of $O^*(55.8^k)$. Recently in [58], an fpt-algorithm of running time $O^*(16^{k+o(k)})$ was presented by further exploring the idea of iteratively partitioning the embedded tree we want to find.

A crown structure is a novel idea that allows us to have powerful reduction rules. Its applications have been wide and successful, which includes a linear-size kernel for the vertex cover problem [32, 52]. We propose a kernelization for Directed k-Internal exploiting a crown structure of the instance, which yields a kernel with $O(k^2)$ vertices. For undirected graphs, Fomin et al. exhibited a $3k$-vertex kernel in [56] and obtained $O^*(8^k)$-algorithm for k-Internal as a corollary.

1.3 Parameterized Algorithms for Constraint Satisfaction Problems

The Constraint Satisfaction Problems (CSPs) form one of the most important combinatorial problems as they offer a natural language to formulate a huge variety of combinatorial problems with Coloring, Max-Cut and Satisfiability as notable examples. In this thesis, we focus on the CSPs in which we want to maximize the number of satisfied constraints. Since a majority of CSPs of this type allow randomized α-approximation algorithms by which the number of satisfied constraints have a tight lower bound, they are often trivially FPT under the standard parameterization. Here, a lower bound *tight* in the sense that it is optimal for an infinite sequence of instances. Consider the following illustration.

Given a digraph $D = (V, A)$, find an acyclic subdigraph of D with the maximum number of arcs. A standard parameterization for this problem asks whether D contains an acyclic subdigraph with at least k arcs. It is easy to prove that this parameterized problem is fixed-parameter tractable by observing that D always has an acyclic subdigraph with at least $|A|/2$ arcs. Indeed, consider a bijection $\alpha : V \to \{1, \dots, |V|\}$ and the following subdigraphs of D: $(V, \{ xy \in A : \alpha(x) < \alpha(y) \})$ and $(V, \{ xy \in A : \alpha(x) > \alpha(y) \})$. Both subdigraphs are acyclic and at least one of them has at least $|A|/2$ arcs. However, $k \leq |A|/2$ for every small value of k and almost every practical value of $|A|$ and, thus, our standard parameterization is of almost no practical or theoretical interest.

Instead, one can consider the following parameterized problem: decide whether $D = (V, A)$ contains an acyclic subdigraph with at least $|A|/2 + k$ arcs. We choose $|A|/2 + k$ because $|A|/2$ is a *tight lower bound* on the size of a largest acyclic subdigraph. Indeed, the size of a largest acyclic subdigraph of a symmetric digraph $D = (V, A)$ is precisely $|A|/2$. (A digraph $D = (V, A)$ is *symmetric,* if $xy \in A$ implies $yx \in A$.)

Parameterizations *above a guaranteed value* were first considered by Mahajan and Raman [90] for the problems Max-Sat and Max-Cut. They devised an algorithm for Max-Sat with running time $O^*(1.618^k + \sum_{i=1}^{m} |C_i|)$ that finds, for a multiset $\{C_1, \ldots, C_m\}$ of m clauses, a truth assignment satisfying at least $\lceil m/2 \rceil + k$ clauses, or decides that no such truth assignment exists ($|C_i|$ denotes the number of literals in C_i).

In a recent paper [91], Mahajan, Raman and Sikdar provided several examples of problems of this type and argued that a natural parameterization is one above a tight lower bound for maximization problems, and below a tight upper bound for minimization problems. Furthermore, they observed that only a few non-trivial results are known for problems parameterized above a tight lower bound [69, 71, 105, 90], and they listed several problems parameterized above a tight lower bound whose complexity is unknown. The difficulty in showing whether such a problem is fixed-parameter tractable can be illustrated by the fact that often we even do not know whether the problem is in XP, i.e., can be solved in time $O(|I|^{g(k)})$ for a computable function $g(k)$. For example, it is non-trivial to see that the above-mentioned digraph problem is in XP when parameterized above the $|A|/2$ bound.

In this thesis, we introduce the Strictly Above/Below Expectation Method (SABEM), a novel approach for establishing the fixed-parameter tractability of maximization problems parameterized above tight lower bounds and minimization problems parameterized below tight upper bounds. The new method is based on probabilistic arguments and utilizes certain probabilistic inequalities. This method can be seen as a tool for exhibiting the existence of a kernel. A detailed account of SABEM and its application to a number of new and open constraint satisfaction problems will be provided in Chapter 6.

In Section 6.2, we consider the Linear Ordering problem, a generalization of the problem discussed above: Given a digraph $D = (V, A)$ in which each arc ij has a positive integral weight w_{ij}, find an acyclic subdigraph of D of maximum weight. Observe that $W/2$, where W is the sum of all arc weights, is a tight lower bound for Linear Ordering. We prove that the problem parameterized above $W/2$ is fixed-parameter tractable and admits a quadratic kernel. This parameterized problem generalizes the parameterized maximum acyclic subdigraph problem stated as open in [91].

In Section 6.3, we consider the problem Max Lin-2: Given a system of m linear equations $e_1, \ldots, e_m$ in n variables over GF(2), and for each equation e_j a positive integral weight w_j; find an assignment of values to the n variables that maximizes the total weight of the satisfied equations. Various algorithmic aspects of Max Lin have been well-studied (cf. [6, 73, 74]). Perhaps, the best known result on Max Lin is the following inapproximability theorem of Håstad [73]: unless P=NP, for each $\varepsilon > 0$ there is no polynomial time algorithm for distinguishing instances of Max 3-Lin-2 in which at least $(1-\varepsilon)m$ equations can be simultaneously satisfied from instances in which less than $(1/2+\varepsilon)m$ equations can be simultaneously satisfied. It is not difficult to see that $W/2$, where $W = w_1 + \cdots + w_m$, is a tight lower bound for Max Lin-2. The complexity of the problem parameterized above $W/2$ is open [91]. We prove that for three nontrivial special cases there exist kernels with $O(k^2)$ variables and equations. We also show that if we allow the weights w_j to be positive reals, the problem is NP-hard already if $k = 1$ and each equation involves two variables.

In Section 6.4, we explore the problem Ordinal Embeddings or Betweenness. The problem of mapping points with measured pairwise distances into a target metric space has a long history and been studied extensively from multiple perspectives due to its numerous applications. The quality of such an embedding can be measured with various objectives; for example isometric embeddings preserve all distances while aiming at low-dimensional target spaces. Yet, for many contexts in nearest-neighbor search, visualization, clustering and compression it is the order of distances rather than the distances themselves that captures the relevant information. The study of such Ordinal Embeddings dates back to the 1950's and has recently witnessed a surge in interest [1, 12, 18, 78]. In an ordinal embedding the relative order between pairs of distances must be preserved as much as possible, i.e., one minimizes the relaxation of an ordinal embedding defined as the maximum ratio between two distances whose relative order is inverted by the embedding. Such linear arrangements are of significant interest in molecular biology, where for example markers on a chromosome need to be totally ordered as to satisfy the maximum number of constraints [38, 63]. More theoretical interest comes from the constraint programming framework with unbounded domains and interval graph recognition [86].

Already deciding if *all* constraints can be satisfied by some linear arrangement is an NP-complete problem [99]. Therefore, the complementary question of whether all but k constraints are satisfiable by some linear arrangement is not fixed-parameter tractable, unless $\mathsf{P} = \mathsf{NP}$. On the other hand, any uniformly random permutation of the variables satisfies at least one-third of all constraints, and this fraction is tight. Better approximation ratios are hard to achieve: the fraction of one-third is best-possible under the Unique Games Conjecture [30], and it is NP-hard to find a linear arrangement that satisfies a

$1 - \varepsilon$ fraction of the constraints for some $\varepsilon \in (0, 1/48)$ [33]. The mere positive result is a polynomial time algorithm that either determines that there is no linear arrangement that satisfies all m constraints or finds a linear arrangement satisfying at least half of them [33, 92].

So the right question to ask is whether there exists a linear arrangement that satisfies at least $|C|/3 + k$ of the constraints. The parameterized complexity of this problem attributed to Benny Chor was open, and was stated as such in [94]. We show that Betweenness parameterized above the tight lower bound has a kernel of quadratic size, namely, any instance is polynomial-time reducible to an equivalent instance of size $O(k^2)$.

The Maximum r-Satisfiability Problem (Max-r-Sat) from Section 6.5 is a classic optimization problem with a wide range of real-world applications. The task is to find a truth assignment to a multiset of clauses, each with exactly r literals, that satisfies as many clauses as possible, or in the decision version of the problem, to satisfy at least t clauses where t is given with the input. Even Max-2-Sat is NP-hard [61] and APX-hard [73], in strong contrast with 2-Sat which is solvable in linear time [10].

It is always possible to satisfy a $1 - 2^{-r}$ fraction of a given multiset of clauses with exactly r literals each and this lower bound is tight. Using SABEM we show that for every fixed r we can decide in time $O(m) + 2^{O(k^2)}$ whether a given multiset of m clauses admits a truth assignment that satisfies at least $((2^r - 1)m + k)/2^r$ clauses. This answers a question posed by Mahajan, Raman and Sikdar [91].

1.4 Thesis Outline and Bibliographic Note

The remainder of the thesis is organized as follows. Chapter 2 provides the background notion for the study. We consider three problems on digraphs in Part I. Chapter 3 pursue the study of the parameterized Directed Maximum Leaf. We consider k-Out-Tree in Chapter 4 and examine the problem Directed Minimum Leaf from the parameterized perspective in Chapter 5. In Part II we consider several constraint satisfaction problems parameterized above their tight bounds. We present a new method of kernelization for such parameterization in Chapter 6 and describe how the method can be used in concrete problems. Other problem-specific combinatorial approaches will be examined in Chapter 7.

All the results in this thesis are new unless otherwise specified. Most of the results have been presented at journals or conferences. Below is a list of the previous articles that our work is based on.

- [70] Minimum leaf out-branching and related problems, joint work with G. Gutin, I. Razgon, *Theoretical Computer Science* 410 (45), pp. 4571-4579 (2009).

- [43] On the Complexity of Minimum Leaf Out-branching Problem, joint work with P. Dunkelmann, G. Gutin, *Discrete Applied Mathematics* 157 (13), pp. 3000-3004 (2009).

- [41] FPT Algorithms and Kernels for the Directed k-Leaf Problem, joint work with J. Daligault, G. Gutin, A. Yeo, *Journal of Computer and System Sciences* 76 (2), pp. 144-152 (2010).

- [36] Algorithm for Finding k-Vertex Out-trees and its Application to k-Internal Out-branching Problem, joint work with N. Cohen, F. V. Fomin, G. Gutin, S. Saurabh, A. Yeo, To appear in *Journal of Computer and System Sciences*.

- [68] Probabilistic Approach To Problems Parameterized Above Tight Lower Bound, joint work with G. Gutin, S. Szeider, A. Yeo, *Proceedings of the 4th International Workshop on Parameterized and Exact Computation* (IWPEC 2009).

- [5] Solving Max-r-SAT Above a Tight Lower Bound, joint work with N. Alon, G. Gutin, S. Szeider, A. Yeo, *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms* (SODA 2010).

- [67] Betweenness Parameterized Above Tight Lower Bound, joint work with G. Gutin, M. Mnich, A. Yeo, To appear in *Journal of Computer and System Sciences*.

- [40] Lower Bounds for Maxima of Functions with Boolean Variables and Max Lin Above Average Problem, joint work with R. Crowston, G. Gutin, M. Jones, I.Z. Ruzsa, To appear in *the 12th Scandinavian Symposium and Workshop on Algorithm Theory* (SWAT 2010).

Chapter 2

Notions

In this chapter we introduce some basic notions and facts that are relevant to our work.

2.1 Graph Theory

We deal with both undirected and directed graphs in the following chapters. Also we make use of some width measures of a (directed) graph such as tree-width, directed tree-width and DAG-width.

Undirected graphs: An *undirected graph* G consists of a set $V(G)$ and a set $E(G)$ which is a subset of all 2-sets of $V(G)$. The elements of $V(G)$ are *vertices* and the elements of $E(G)$ are *edges*. We only consider finite graphs, that is, $V(G)$ and $E(G)$ are finite and by definition we restrict ourselves to graphs without parallel edges and loops. The number of vertices and edges of the graph under consideration will be denoted n and m respectively. An edge $e = \{u, v\} \in E(G)$ have the elements u,v as the *endpoints* and is denoted as (u, v) or uv whichever is convenient. The two endpoints u, v of an edge $e = (u, v)$ are said to be *adjacent* with each other and are *incident* with the edge e. Likewise the edge e is incident with u and v.

The set $\{u \in V(G) : (u, v) \in E(G)\}$ of vertices adjacent to a vertex $v \in V(G)$ is called a *neighborhood*, or sometimes an open neighborhood, of v and is denoted by $N(v)$. By taking $N(v) \cup \{v\}$, we get a *closed neighborhood* of v. In general, for a set of vertices $X \subseteq V(G)$, the (open) neighborhood of X is the set $\bigcup_{x \in X} N(x) \setminus X$.

The *degree* $d(v)$ of a vertex $v \in V(G)$ is the number of edges incident with v. A vertex of degree 0 is said to be *isolated*.

Let $G' = (V', E')$ and $G = (V, E)$ be are two graphs. If $V' \subseteq V$ and $E' \subseteq E$, then G' is a *subgraph* of G, written as $G' \subseteq G$. If, moreover, G' contains *all* edges $uv \in E$ for all $u, v \in V'$, then G' is an *induced subgraph* of G and written as $G' := G[V']$. A *spanning* subgraph $G' \subseteq G$ is a subgraph of G with $V' = V$. Finally, we say G and G' are *isomorphic*, denote as $G \simeq G'$, if there exists a bijection $\varphi : V \mapsto V'$ such that $uv \in E$ if and only if $\varphi(u)\varphi(v) \in E'$.

A *walk* W in a graph G is an alternating sequence $v_0, e_0, v_1, e_1, \dots, e_{l-1}, v_l$ with $e_i = \{v_i, v_{i+1}\}$ for $0 \leq i < l$. The walk W is usually written as $v_0v_1 \dots v_l$. We say that W is a walk from v_0 to v_l, or a $v_0 - v_l$ walk. The number l of edges in W is the *length* of W. If W is restricted to have all distinct vertices (edges, respectively), it is a *path* (a *trail*, respectively). A walk $W = v_0 \dots v_l$ is said to be a *cycle* if $v_0 = v_l$ and all vertices v_i, $0 \leq i < l$, are distinct. This cycle is denoted by $v_1 \dots v_l (= v_0)$. A cycle with l vertices is denoted as l-cycle.

A graph G is said to be *connected* if there is a path from between every pair of vertices. A maximal connected subgraph of G is a *(connected) component* of G. Any graph is either connected or uniquely decomposed into more than one components.

For further notions in graph theory we refer the readers to the standard textbook by Diestel [46].

Directed graphs: If we take ordered pairs of vertices as the edges in replacement of 2-sets, we obtain a *directed graph D* or a *digraph* in short. For a directed graph D, an ordered pair $e = (u, v)$ is called an *arc* or a *directed edge*, and u,v are the *tail* and *head* of the arc e respectively. An arc $e = (u, v)$ is incident *from* its tail u and incident *to* its head v. The set of vertices and arcs of a digraph D will be denoted by $V(D)$ and $A(D)$ respectively. A digraph is called an *oriented graph* if it has no directed 2-cycle. Most notions defined on undirected graphs are naturally extended to cover directed graphs.

For a vertex v of a subgraph H of a digraph D, $N^+_H(v)$ denotes the set of out-neighbors of v. Also, let $A^+_H(v) = \{vu : \ u \in N^+_H(v)\}$ and let $d^+_H(v) = |N^+_H(v)|$ denote the *outdegree* of v.. The notations $N^-_H(v)$, $A^-_H(v)$ and $d^-_H(v)$ are defined analogously for in-neighbors of v. When $H = D$ we will frequently omit the subscripts in the notation above.

A directed graph D is said to be *strongly connected* if there is a *directed path* from u to v for every ordered pair u, v of $V(D)$. A maximal strongly connected subgraph of D is a *strong (connected) component* of D. The *underlying graph $UG(D)$* of a directed graph D is the undirected graph obtained from D by disregarding the order of elements in every arc and deleting one edge in each pair of parallel edges. A directed graph D is *connected* if $UG(D)$ is connected. The *components* of D are defined as the components of $UG(D)$.

We say that a subgraph T of a digraph D is an *out-tree* if T is an oriented tree with only one vertex r of in-degree zero (called *the root*). We may want to emphasis the root of an out-tree T by saying that T is an out-tree *rooted at r*. The vertices of T of out-degree zero are called *leaves* and all other vertices *internal vertices*. If T is a spanning out-tree, i.e. $V(T) = V(D)$, then T is called an *out-branching* of D. It is easy to decide whether a digraph contains an out-branching.

Lemma 2.1.1. *[13] A digraph D has an out-branching rooted at vertex $r \in V(D)$ if and only if D has a unique strong connectivity component S of D without incoming arcs and $r \in S$. One can check whether D has a unique strong connectivity component and find one, if it exists, in time $O(m+n)$, where n and m are the number of vertices and arcs in D, respectively.*

The monograph by Bang-Jensen and Gutin [13] is a comprehensive source for concepts and results on directed graph.

Tree-width: A *tree decomposition* of an undirected graph G is a pair (X, T) where T is a tree whose vertices we will call *nodes* and $X = \{X_i : i \in V(T)\}$ is a collection of subsets of $V(G)$ (called *bags*) such that

1. $\bigcup_{i \in V(T)} X_i = V(G)$,
2. for each edge $(v, w) \in E(G)$, there is an $i \in V(T)$ such that $v, w \in X_i$, and
3. for each $v \in V(G)$ the set of nodes $\{i : v \in X_i\}$ form a subtree of T.

The *width* of a tree decomposition $(\{X_i : i \in V(T)\}, U)$ equals $\max_{i \in V(T)}\{|X_i| - 1\}$. The *treewidth* of a graph G is the minimum width over all tree decompositions of G. We use the notation $\text{tw}(G)$ to denote the treewidth of a graph G.

By a *tree decomposition of a digraph D* we will mean a tree decomposition of the underlying graph $UG(D)$. Also, $\text{tw}(D) = \text{tw}(UG(D))$.

Sometimes it is convenient to work with a *nice tree decomposition.* Any tree decomposition can be converted into a nice tree decomposition of the same width in linear time, see [94]. A tree decomposition (X, T) of an undirected graph G is *nice* if it is a rooted binary tree and any node $i \in V(T)$ is one of the four types:

1. Leaf Node
2. Join Node: Node i has two children j and k and $X_i = X_j = X_k$.
3. Introduce Node: Node i has a single child j and $X_i = X_j \cup \{u\}$ for some $u \in V(G)$.
4. Forget Node: Node i has a single child j and $X_i = X_j - \{u\}$ for some $u \in V(G)$.

DAG-width: DAG-width was introduced independently by Berwanger et al. [16] and Obdrzalek [96]. A *DAG-decomposition* of a digraph D is a pair (H, χ) where H is an acyclic digraph and $\chi = \{W_h : h \in V(H)\}$ is a family of subsets (called *bags*) of $V(D)$ such that

1. $V(D) = \bigcup_{h \in V(H)} W_h$

2. if $(u, v) \in A(D)$, then there exist $h_1, h_2 \in V(H)$ (it is possible that $h_1 = h_2$) such that $u \in W_{h_1},\ v \in W_{h_2}$ and there is a directed (h_1, h_2)-path in H

3. for all $h, h', h'' \in V(H)$, if h' lies on a directed path from h to h'', then $W_h \cap W_{h''} \subseteq W_{h'}$.

The *width* of a DAG-decomposition (H, χ) is $\max_{h \in V(H)} |W_h| - 1$. The *DAG-width* of a digraph D (dagw(D)) is the minimum width over all possible DAG-decompositions of D.

Directed path-width: A *directed path decomposition* [14] is a special case of DAG-decomposition when H is a directed path. The *directed path-width* of a digraph D (dpw(D)) is defined as the DAG-width above, but DAG-decompositions are replaced by directed path decompositions.

Directed tree-width: Directed tree-width was introduced by Johnson, Robertson, Seymour and Thomas [76]. Let Z be a set of vertices of a digraph D. A set $S \subseteq V(D) - Z$ is *Z-normal* if every directed walk that leaves and again enters S must traverse a vertex of Z. For vertices r, r' of an out-tree T we write $r \leq r'$ if there is a path from r to r' or $r = r'$. An *arboreal decomposition* of a digraph D is a triple (R, X, W), where R is an out-tree (not a subgraph of D), $X = \{X_e : e \in A(R)\}$ and $W = \{W_r : r \in V(R)\}$ are families of sets of vertices of D that satisfy two conditions: (1) $\{W_r : r \in V(R)\}$ is a partition of $V(D)$ into nonempty sets, and (2) for each $e = (r', r'') \in A(R)$ the set $\bigcup\{W_r : r \in V(R), r \geq r''\}$ is X_e-normal. The *width* of (R, X, W) is the least integer w such that for all $r \in V(R)$, $|W_r \cup \bigcup_{e \sim r} X_e| \leq w + 1$, where $e \sim r$ means that r is head or tail of e. The *directed tree-width* of D, dtw(D), is the least integer w such that D has an arboreal decomposition of width w.

2.2 Constraint Satisfaction Problems

A large number of combinatorial problems can be formulated as *constraint satisfaction problems*. In the second part of the thesis, we consider a set of problems which belong to a wide family called the *constraint satisfaction problems (CSPs)*.

A constraint satisfaction problem is a triple $(\mathcal{X}, \mathcal{D}, \mathcal{C})$, where $\mathcal{X}$ is a set of variables, $\mathcal{D}$ the domain for the variables and $\mathcal{C}$ a set of constraints. Each constraint $C \in \mathcal{C}$ consists of a pair (s, R); s is a tuple of k variables called the *scope* and R is a k-ary relation over $\mathcal{D}$. Each relation R in a constraint (s, R) represents the set of all possible combination of value assignments to the variables in the scope s. A solution of a CSP instance $(\mathcal{X}, \mathcal{D}, \mathcal{C})$

is an assignment of values to the variables; $v : \mathcal{X} \to \mathcal{D}$ such that $(v(x_1^s), \ldots, v(x_k^s)) \in R$ for every constraint $C = (s, R) \in \mathcal{C}$ with $s = (x_1^s, \ldots, x_k^s)$.

Example 2.2.1. *The* 3-Coloring *can be formulated as a CSP. Let G be the input graph of* 3-Coloring *and* $\{RED, BLU, YEL\}$ *be the three colors in the palette. We construct a triple* $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ *as:* $\mathcal{X} = V(G)$, $\mathcal{D} = \{RED, BLU, YEL\}$ *and for every edge* $e = (u, v) \in E(G)$ *we have the corresponding constraint* $C_e = (e, R)$, *where* $R = \{RED, BLU, YEL\}^3 \setminus \{(RED, RED), (BLU, BLU), (YEL, YEL)\}$.

Taking a relation as a constraint and strictly confining the combinations of values to the scope of the constraint is a classical framework of CSP and here we are interested in feasibility question. Such constraints are said to be *crisp* in contrast to *soft* constraints [35]. Soft constraints naturally extend the standard notion of CSPs by generalizing the k-ary relations into cost functions of arity k. In this way, we are able to assess the desirability of having a certain combination of values for a set of variables instead of enforcing a certain combination of values to those variables.

In order to obtain the soft CSPs, we substitute each relation R of a k-ary constraint (s, R) in the crisp CSPs with a cost function $\phi : \mathcal{D}^k \mapsto \Omega$, where Ω is a valuation structure representing costs (see [35] for details). The objective is to find an assignment v which minimizes the cost which is the sum of $\phi(v(x_1^s), \ldots, v(x_{|s|}^s))$ over all soft contraints $C = (s, \phi) \in \mathcal{C}$.

Example 2.2.2. *A standard CSP is a special case of soft CSP. To see this, we take* $\Omega = \{0, \infty\}$ *and for each constraint* $(s, R) \in \mathcal{C}$ *we construct a constraint* (s, ϕ_R) *such that* $\phi_R(d_1, \ldots, d_{|s|}) = 0$ *if and only if* $(d_1, \ldots, d_{|s|}) \in R$. *A standard CSP instance is a yes-instance if and only if the constructed soft CSP instance has a solution of finite cost.*

In this thesis, we will consider various soft CSPs and they will be referred to simply as CSPs.

2.3 Probabilistic Method

The probabilistic method is one of the most interesting and useful approaches as an algorithmic tool as well as a proof technique. In principle, a positive probability $\mathbb{P}(A) > 0$ for some event A means that there is at least one point in the probability space which belongs to A. Typically one defines an appropriate probability space and tries to prove that the desired *good* event X takes place with positive probability, which implies the *existence* of a structure (i.e. point in the probability space) with the desired property (i.e. belonging to the good event).

Example 2.3.1. *Ramsey's theorem says that, given an integer $k \geq 0$, every large enough (i.e. as large as the Ramsey number $R(k)$) graph) graph G contains either a clique or an independent set on k vertices. There is a simple probabilistic argument stating that $R(k)$ should be greater than $2^{k/2}$.*

We color each edge of K_n by color 0 or 1 independently at random. Color 0 and 1 can be understood as deleting and preserving the edge, respectively. The set of instances of all possible colorings form the probability space. We define the event A_S as a set of colored instances in which the induced subgraph of K_n on S is monochromatic, that is, either a clique or an independent set. A bad *event in this case is the set of colored instances which belong to none of A_S's. Simple computation show that if $n \leq 2^{k/2}$ the probability of the* bad *event is strictly positive, which implies the existence of a colored instance with the undesired property. Therefore the Ramsey number $R(k)$ should be greater than $2^{k/2}$.*

Linearity of expectation: Given random variables $X_1, \ldots, X_n$, the linearity of expectation states that

$$\mathbb{E}(X_1 + \ldots + X_n) = \mathbb{E}(X_1) + \ldots + \mathbb{E}(X_n).$$

This property allows one to compute the expectation of a variable by decomposing it into a linear combination of simpler variables.

Averaging argument: The averaging argument utilizes the fact that there is a point for which $X \geq \mathbb{E}(X)$ and a point for which $X \leq \mathbb{E}(X)$ in the probability space.

For further reading on the probabilistic method, we refer the reader to the textbook [7] by Alon and Spencer.

2.4 Parameterized Complexity

We recall some basic notions of parameterized complexity here. The bibliography is enormous. For a start, we refer the reader to the established monographs [48, 55, 94].

A problem is a language Q over a finite alphabet Σ. A string $x \in \Sigma^*$, called an *instance* of the problem Q, is a *yes-instance* if $x \in Q$ and it is a *no-instance* otherwise. In classical decision problems, we are interested in determining whether a given instance $x \in \Sigma^*$ is a yes-instance or no-instance.

In parameterized complexity we introduce *parameterization* of Σ^*, which is a mapping κ from Σ^* into $\mathbb{N}$. A *parameterized* problem is a language $\Pi \subseteq \Sigma^* \times \mathbb{N}$ with $k := \kappa(x)$ for each instance $(x, k) \subseteq \Sigma^* \times \mathbb{N}$. The second element of a parameterized instance is called the *parameter*. A yes-instance and a no-instance is defined likewise. The study of the membership problem for parameterized languages and complexity analysis within parameterized framework lies at the heart of parameterized complexity theory.

We say that Π is *fixed-parameter tractable (FPT)*, if membership of (x, k) in Π can be decided in time $O(f(k)|x|^{O}(1))$ for some computable function $f(k)$ independent of $|x|$. Let Π be a parameterized problem. A *reduction R to a problem kernel* (or *kernelization*) is a many-to-one transformation from $(x, k) \in \Pi$ to $(x', k') \in \Pi$ such that (i) $(x, k) \in \Pi$ if and only if $(x', k') \in \Pi$, (ii) $k' \leq k$ and $|x'| \leq g(k)$ for some function g and (iii) R is computable in time polynomial in $|x|$ and k. In kernelization, an instance (x, k) is reduced to another instance (x', k'), which is called the *problem kernel* or simply kernel; $|x'|$ is the *size* of the kernel.

It is easy to see that a decidable parameterized problem is FPT if and only if it admits a kernelization (cf. [55, 94]); however, the problem kernels obtained by this general result have impractically large size. Therefore, one tries to develop kernelizations that yield problem kernels of smaller size. The survey of Guo and Niedermeier [65] on kernelization lists some problem for which polynomial size kernels and exponential size kernels were obtained.

As a more generalized form of kernelization, recently the notion of *bikernelization* has been introduced in [5]. A bikernelization from L to L' is of interest especially when L' is a well-studied problem.

Given a pair L, L' of parameterized problems, a *bikernelization from L to L′* is a polynomial-time algorithm that maps an instance (x, k) to an instance (x', k') (the *bikernel*) such that (i) $(x, k) \in L$ if and only if $(x', k') \in L'$, (ii) $k' \leq f(k)$, and (iii) $|x'| \leq g(k)$ for some functions f and g. The function $g(k)$ is called the *size* of the bikernel. Observe that a *kernelization* of a parameterized problem L is simply a bikernelization from L to itself, i.e., a bikerenelization generalizes a kernelization.

Recall that a decidable parameterized problem is fixed-parameter tractable if and only if it admits a kernelization. This result can be extended as follows: A decidable parameterized problem L is fixed-parameter tractable if and only if it admits a bikernelization from itself to a decidable parameterized problem L'. Indeed, if L is fixed-parameter tractable, then L is decidable and admits a bikernelization to itself. If L is decidable and admits a bikernelization from itself to a parameterized problem L', then (x, k) can be decided by first mapping it to (x', k') in polynomial time and then deciding (x', k') in time depending only on k', and thus only on k.

We are especially interested in cases when kernels are of polynomial size. The next lemma is similar to Theorem 3 in [24]. We repeat the proof for completeness.

Lemma 2.4.1. *Let L, L' be a pair of decidable parameterized problems such that L' is in NP, and L is NP-complete. If there is a bikernelization from L to L' producing a bikernel of polynomial size, then L has a polynomial-size kernel.*

Proof. Consider a bikernelization from L to L' that maps an instance $(x, k) \in L$ to an instance $(x', k') \in L'$ with $k' \leq f(k)$. Since L' is in NP and L is NP-complete, there exists a polynomial time reduction from L' to L. Thus, we can find in polynomial time an instance (x'', k'') of L which is decision-equivalent with (x', k'), and in turn with (x, k). Observe that $|x''| \leq |x'|^{O(1)} \leq k^{O(1)}$ and $k'' \leq (k')^{O(1)} + (|x'|)^{O(1)} \leq f(k)^{O(1)} + k^{O(1)}$. Thus, (x'', k'') is a kernel of L of polynomial size. □

Part I

Parameterized Algorithms on Digraph Problems

Chapter 3

Directed Maximum Leaf Problem

The Maximum Leaf problem is an optimization problem to find a spanning tree with the maximum number of leaves in a given undirected graph G. Its natural extension on directed graphs is the Directed Maximum Leaf problem, which is to find an out-branching with the maximum number of leaves in an input digraph. In this chapter we study the parameterized version of the Directed Maximum Leaf problem called the Directed k-Leaf: given a digraph D and an integral parameter k, decide whether D has an out-branching with at least k leaves. If we add a condition that every out-branching in Directed k-Leaf must be rooted at a given vertex r, we obtain a variation of Directed k-Leaf called the Rooted Directed k-Leaf problem.

In this chapter, we present a fixed-parameter algorithm for Directed k-Leaf. Our algorithm runs in time $O^*(3.72^k)$. We also obtain a linear size kernel for Directed k-Leaf restricted to acyclic digraphs. Notice that (i) Directed Max Leaf restricted to acyclic digraphs is still NP-hard [4], and (ii) for acyclic digraphs Directed k-Leaf and Rooted Directed k-Leaf are equivalent since all out-branchings must be rooted at the unique vertex of in-degree zero.

Let D be a digraph, T an out-tree and $L \subseteq V(D)$. A (T, L)-*out-tree* of D is an out-tree T' of D such that (1) $A(T) \subseteq A(T')$, (2) L are leaves in T', (3) T and T' have the same root. A (T, L)-*out-branching* is a (T, L)-out-tree which is spanning. Let $\ell_{\max}(D, T, L)$ be the maximum number of leaves over all (T, L)-out-branchings of D. We set this number to 0 if there is no (T, L)-out-branching. For an out-tree T in a digraph D, $\mathrm{Leaf}(T)$ denotes the set of leaves in T and $\mathrm{Int}(T) = V(T) - \mathrm{Leaf}(T)$, the set of *internal vertices* of T. For any vertex x in a tree T let T_x denote the maximal subtree of T which has x as its root.

Throughout this chapter we use a triple (D, T, L) to denote a given digraph D, an out-tree T of D and a set of vertices $L \subseteq V(D) - \mathrm{Int}(T)$. We denote by $\hat{D}(T, L)$ the subgraph of D obtained after deleting all arcs out of vertices in L and all arcs not in $A(T)$ which go into a vertex in $V(T)$. When T and L are clear from the context we will omit them and denote $\hat{D}(T, L)$ by $\hat{D}$. We restate Lemma 2.1.1 introduced in Chapter 2 as it will be used in the rest of the chapter.

Reminder of Lemma 2.1.1 *A digraph D has an out-branching if and only if D has a single strong component without incoming arcs. One can decide whether a digraph has an out-branching in time $O(n + m)$.*

3.1 $O^*(4^k)$ Time Algorithm

In this section we present a slightly modified version of the algorithm in [82]. Our algorithm differs from that in [82] as follows. We decide in an earlier stage which one of the current leaves of T cannot be a leaf in a final (T, L)-out-branching and make them to be internal vertices based on Lemma 3.1.3, see the while-loop in lines 2-4 in Algorithm $\mathcal{A}(D, T, L)$. This decision works as a preprocessing of the given instance and gives us a better chance to come up with a (T, L)-out-tree with at least k leaves more quickly. A more important reason for this step is the fact that our algorithm is easier than the main algorithm in [82] to transform into a faster algorithm.

The following is a folklore, and its proof can be found in [82].

Lemma 3.1.1. *If there is an out-branching rooted at vertex r, whenever we have an out-tree rooted at r with at least k leaves we can extend it to an out-branching rooted at r with at least k leaves in time $O(m + n)$.*

Lemma 3.1.2. *Given a triple (D, T, L), we have $\ell_{\max}(D, T, L) = \ell_{\max}(\hat{D}, T, L)$.*

Proof. If there is no (T, L)-out-branching in D, the subgraph $\hat{D}$ does not have a (T, L)-out-branching either and the equality holds trivially. Hence suppose that T^* is a (T, L)-out-branching in D with $\ell_{\max}(D, T, L)$ leaves. Obviously we have $\ell_{\max}(D, T, L) \geq \ell_{\max}(\hat{D}, T, L)$. Since the vertices of L are leaves in T^*, all arcs out of vertices in L do not appear in T^*, i.e. $A(T^*) \subseteq A(D) \setminus \{A^+_D(x) : x \in L\}$. Moreover $A(T) \subseteq A(T^*)$ and thus all arcs not in $A(T)$ which go into a vertex in $V(T)$ do not appear in T^* since otherwise we have a vertex in $V(T)$ with more than one arc of T^* going into it (or, the root has an arc going into it). Hence we have $A(T^*) \subseteq A(\hat{D})$ and the above equality holds. □

Lemma 3.1.3. *Given a triple (D, T, L), the following equality holds for each leaf x of T.*

$$\ell_{\max}(D, T, L) = \max\{\ell_{\max}(D, T, L \cup \{x\}), \ell_{\max}(D, T \cup A^+_{\hat{D}}(x), L)\}$$

Proof. If $\ell_{\max}(D, T, L) = 0$ then the equality trivially holds, so we assume that $\ell_{\max}(D, T, L) \geq 1$. Since any $(T, L \cup \{x\})$-out-branching or $(T \cup A^+_{\hat{D}}(x), L)$-out-branching is a (T, L)-out-branching as well, the inequality $\geq$ obviously holds. To show the opposite direction, suppose T' is an optimal (T, L)-out-branching. If x is a leaf in T', then T' is a $(T, L \cup \{x\})$-out-branching and $\ell_{\max}(D, T, L) \le \ell_{\max}(D, T, L \cup \{x\})$.

Suppose x is not a leaf in T'. Delete all arcs entering $N^+_{\hat{D}}(x)$ in T', add $A^+_{\hat{D}}(x)$ and let T'' denote the resulting subgraph. Note that $d^-_{T''}(y) = 1$ for each vertex y in T'' which is not the root and $A(T'') \subseteq A(\hat{D})$. In order to show that T'' is an out-branching it suffices to see that there is no cycle in T'' containing x. If there is a cycle C containing x in T'' and $xy \in A(C)$, then $C - \{xy\}$ forms a directed (y, x)-path in $\hat{D}$. However this is a contradiction as $x \in V(T)$ and $y \notin V(T)$ and there is no path from $V(D) - V(T)$ to $V(T)$ in $\hat{D}$. Hence T'' is an out-branching.

As no vertex in L has any arcs out of it in $\hat{D}$ we note that $L \subseteq \text{Leaf}(T'')$. Furthermore we note that $A(T) \subseteq A(T'')$ as $A(T) \subseteq A(T')$ and all arcs we deleted from $A(T')$ go to a vertex not in $V(T)$. Therefore T'' is a (T, L)-out-branching which has as many leaves as T'. This shows $\ell_{\max}(D, T, L) \le \ell_{\max}(D, T \cup A^+_{D'}(x), L)$. □

Definition 3.1.4. *Given a triple (D, T, L) and a vertex $x \in Leaf(T) - L$, define $T^{root}_{D,L}(x)$ as follows.*

(1) $x' := x$.

(2) *While $d^+_{\hat{D}}(x') = 1$ add $A^+_{\hat{D}}(x') = \{x'y\}$ to T and let $x' := y$.*

(3) *Add $A^+_{\hat{D}}(x')$ to T.*

Now let $T^{root}_{D,L}(x) = T_x$. That is, $T^{root}_{D,L}(x)$ contains exactly the arcs added by the above process.

The idea behind this definition is the following: during the algorithm, we will decide that a given leaf x of the partial out-tree T built thus far is not a leaf of the out-branching we are looking for. Then adding the out-arcs of x to T is correct. To make sure that the number of leaves of T has increased even when $d^+_{V-V(T)}(x) = 1$, we add $T^{root}(x)$ to T instead of just adding the single out-arc of x, as described in the following.

Lemma 3.1.5. *Suppose we are given a triple (D, T, L) and a leaf $x \in Leaf(T) - L$. If $\ell_{\max}(D, T, L \cup \{x\}) \ge 1$ then the following holds.*

(i) *If $|Leaf(T^{root}_{D,L}(x))| \ge 2$ then $\ell_{\max}(D, T, L) = \max\{\ell_{\max}(D, T, L \cup \{x\}), \ell_{\max}(D, T \cup T^{root}_{D,L}(x), L)$.*

(ii) *If $|Leaf(T^{root}_{D,L}(x))| = 1$ then $\ell_{\max}(D, T, L) = \ell_{\max}(D, T, L \cup \{x\})$.*

Proof. Assume that T' is an optimal (T, L)-out-branching and that $|\text{Leaf}(T'_x)| = 1$. We will now show that $\ell_{\max}(D, T, L \cup \{x\}) = |\text{Leaf}(T')| = \ell_{\max}(D, T, L)$. If x is a leaf of T' then this is clearly the case, so assume that x is not a leaf of T'. Let y be the unique out-neighbor of

x in T'. As $\ell_{\max}(D,T,L\cup\{x\})\geq 1$ we note that there exists a path $P=p_0p_1p_2\dots p_r(=y)$ from the root of T to y in $\hat{D}(T,L\cup\{x\})$. Assume that q is chosen such that $p_q\notin T'_x$ and $\{p_{q+1},p_{q+2},\dots,p_r\}\subseteq V(T'_x)$. Consider the digraph $D^*=D[V(T'_x)\cup\{p_q\}-\{x\}]$ and note that p_q can reach all vertices in D^*. Therefore there exists an out-branching in D^*, say T^*, with p_q as the root. Let T'' be the out-branching obtained from T' by deleting all arcs in T'_x and adding all arcs in T^*. Note that $|\text{Leaf}(T'')|\geq|\text{Leaf}(T')|$ as $\text{Leaf}(T^*)\cup\{x\}$ are leaves in T'' and $\text{Leaf}(T'_x)\cup\{p_q\}$ are the only leaves in T' which may not be leaves in T'' (and $|\text{Leaf}(T'_x)\cup\{p_q\}|=2$). Therefore $\ell_{\max}(D,T,L\cup\{x\})\geq|\text{Leaf}(T')|=\ell_{\max}(D,T,L)$. As we always have $\ell_{\max}(D,T,L)\geq\ell_{\max}(D,T,L\cup\{x\})$ we get the desired equality.

This proves part (ii) of the lemma, as if $|\text{Leaf}(T^{root}_{D,L}(x))|=1$ then any optimal (T,L)-out-branching T', must have $|\text{Leaf}(T'_x)|=1$.

We therefore consider part (i), where $|\text{Leaf}(T^{root}_{D,L}(x))|\geq 2$. Let Q denote the set of leaves of $T^{root}_{D,L}(x)$ and let $R=V(T^{root}_{D,L}(x))-Q$. Note that by the construction of $T^{root}_{D,L}(x)$ the vertices of R can be ordered $(x=)r_1,r_2,\dots,r_i$ such that $r_1r_2\dots,r_i$ is a path in $T^{root}_{D,L}(x)$. As before let T' be an optimal (T,L)-out-branching and note that if any r_j $(1\leq j\leq i)$ is a leaf of T' then $|\text{Leaf}(T'_x)|=1$ and the above gives us $\ell_{\max}(D,T,L\cup\{x\})=\ell_{\max}(D,T,L)$. This proves part (i) in this case, as we always have $\ell_{\max}(D,T,L)\geq\ell_{\max}(D,T\cup T^{root}_{D,L}(x),L)$. Therefore no vertex in $\{r_1,r_2,\dots,r_i\}$ is a leaf of T' and all arcs $(x=)r_1r_2,r_2r_3,\dots,r_{i-1}r_i$ belong to T'. By Lemma 3.1.3 we may furthermore assume that T' contains all the arcs from r_i to vertices in Q. Therefore $T^{root}_{D,L}(x)$ is a subtree of T' and $\ell_{\max}(D,T,L)=\ell_{\max}(D,T\cup T^{root}_{D,L}(x),L)$. This completes the proof of part (i). □

The following is an $O(4^kn^{O(1)})$ algorithm. We perform the procedure $\mathcal{A}(D,\{x\},\emptyset)$ for every vertex $x\in V(D)$. If one of the returns of $\mathcal{A}(D,\{x\},\emptyset)$ is "YES", then the final output is "YES". Otherwise, we output "NO". Its complexity can be obtained similarly to [82]. We restrict ourselves only to proving its correctness.

Algorithm 1 $\mathcal{A}(D,T,L)$

```
if the number of vertices with in-degree 0 differs from 1 then return "NO"
while there is a vertex $x \in \text{Leaf}(T)-L$ such that $\ell_{\max}(D,T,L\cup\{x\})=0$ do
    add the arcs $A^+_{\hat{D}}(x)$ to $T$
end while
if $|L|\geq k$ then return "YES"
if the number of leaves in $T$ is at least $k$ then return "YES"
if all leaves in $T$ belong to $L$ then return "NO"
Choose a vertex $x\in\text{Leaf}(T)-L$.
$B_1 := \mathcal{A}(D,T,L\cup\{x\})$ and $B_2$ :="NO".
if $|\text{Leaf}(T^{root}_{D,L}(x))|\geq 2$ then $B_2 := \mathcal{A}(D,T\cup T^{root}_{D,L}(x),L)$.
if either $B_1$ or $B_2$ is "YES" then return "YES"
else return "NO"
```

Remark 3.1.6. *While line 5 is unnecessary, we keep it since it is needed in the next algorithm where $L \subseteq Leaf(T)$ is not necessarily true, see line 16 in the next algorithm, where $p_0 \notin V(T)$.*

Theorem 3.1.7. *Algorithm $\mathcal{A}(D, T, L)$ works correctly. In other words, D has a (T, L)-out-branching with at least k leaves if and only if Algorithm $\mathcal{A}(D, T, L)$ returns "YES".*

Proof. We begin by showing that a call to $\mathcal{A}(D, T, L)$ is always made with a proper argument (D, T, L), that is, T is an out-tree of D and $L \cap \mathrm{Int}(T) = \emptyset$. Obviously the initial argument $(D, \{x\}, \emptyset)$ is proper. Suppose (D, T, L) is a proper argument. It is easy to see that $(D, T, L \cup \{x\})$ is a proper argument. Let us consider $(D, T \cup T^{root}_{D,L}(x), L)$. By Definition 3.1.4 we note that $T \cup T^{root}_{D,L}(x)$ is an out-tree in D and since we consider the digraph $\hat{D}$ at each step in Definition 3.1.4 we note that no vertex in L is an internal vertex of $T \cup T^{root}_{D,L}(x)$. Hence $(D, T \cup T^{root}_{D,L}(x), L)$ is a proper argument.

Consider the search tree ST that we obtain by running the algorithm $\mathcal{A}(D, T, L)$. First consider the case when ST consists of a single node. If $\mathcal{A}(D, T, L)$ returns "NO" in line 1, then clearly we do not have a (T, L)-out-branching. The while-loop of lines 2-4 is valid by Lemma 3.1.3, i.e. it does not change the return of $\mathcal{A}(D, T, L)$. So now consider lines 5-7. As $\ell_{\max}(D, T, L) \geq 1$ after line 1, and by Lemma 3.1.3 the value of $\ell_{\max}(D, T, L)$ does not change by the while-loop we note that $\ell_{\max}(D, T, L) \geq 1$ before we perform lines 5-7. Therefore there exists a (T, L)-out-branching in D. If $|L| \geq k$ or $|\mathrm{Leaf}(T)| \geq k$ then, by Lemma 3.1.1, any (T, L)-out-branching in D has at least k leaves and the algorithm returns "YES". If $\mathrm{Leaf}(T) \subseteq L$ then the only (T, L)-out-branching in D is T itself and as $|\mathrm{Leaf}(T)| < k$ the algorithm returns "NO" as it must do. Thus, the theorem holds when ST is just a node.

Now suppose that ST has at least two nodes and the theorem holds for all successors of the root R of ST. By the assumption that R makes further recursive calls, we have $\ell_{\max}(D, T, L) \geq 1$ and there exists a vertex $x \in \mathrm{Leaf}(T) - L$. If there is a (T, L)-out-branching with at least k leaves, then by Lemma 3.1.5 there is a $(T, L \cup \{x\})$-out-branching with at least k leaves or $(T \cup T^{root}_{D,L}(x), L)$-out-branching with at least k leaves. By induction hypothesis, one of B_1 or B_2 is "YES" and thus $\mathcal{A}(D, T, L)$ correctly returns "YES". Else if $\ell_{\max}(D, T, L) < k$, then again by Lemma 3.1.5 and induction hypothesis both B_1 and B_2 are "NO". Therefore the theorem holds for the root R of ST, which completes the proof. □

3.2 Faster FPT-algorithm

We now show how the algorithm from the previous section can be made faster by adding an extra vertex to the set L in certain circumstances. Recall that the while-loop in the above algorithm $\mathcal{A}(D, T, L)$ and in our new algorithm $\mathcal{B}(D, T, L)$ is new compared to the algorithm in [82]. We will also allow L to contain vertices which are not leaves of the current out-tree T. The improved algorithm is given as follows. We perform the procedure $\mathcal{B}(D, \{x\}, \emptyset)$ for every vertex $x \in V(D)$. If one of the returns of $\mathcal{B}(D, \{x\}, \emptyset)$ is ”YES”, then the final output is ”YES”. Otherwise, we output ”NO”.

Algorithm 2 $\mathcal{B}(D, T, L)$

```
1:  if $\ell_{\max}(D, T, L) = 0$ then return ”NO”
2:  while there is a vertex $x \in \text{Leaf}(T) - L$ such that $\ell_{\max}(D, T, L \cup \{x\}) = 0$ do
3:      add the arcs $A^+_{\hat{D}}(x)$ to $T$
4:  end while
5:  if $|L| \geq k$ then return ”YES”
6:  if the number of leaves in $T$ is at least $k$ then return ”YES”
7:  if all leaves in $T$ belong to $L$ then return ”NO”
8:  Choose a vertex $x \in \text{Leaf}(T) - L$. Color $x$ red and let $H_x := \hat{D}$.
9:  Let $z$ be the nearest ancestor of $x$ in $T$ colored red, if it exists.
10: $L' := L \cup \{x\}$.
11: if $z$ exists and $T_z$ has exactly two leaves $x$ and $x'$ and $x' \in L$ then
12:     Let $P = p_0p_1 \dots p_r$ be a path in $H_z - A^+_{\hat{D}}(z)$ such that $V(P) - V(T_z) = \{p_0\}$ and
        $p_r \in N^+_{\hat{D}}(z)$
13:     $L' := L \cup \{p_0, x\}$
14: end if
15: $B_1 := \mathcal{B}(D, T, L')$ and $B_2 :=$”NO”
16: if $|\text{Leaf}(T^{root}_{D,L}(x))| \geq 2$ then $B_2 := \mathcal{B}(D, T \cup T^{root}_{D,L}(x), L)$.
17: if either $B_1$ or $B_2$ is ”YES” then return ”YES”
18: else return ”NO”
```

The existence of P in line 12 follows from the fact that z was colored red, hence adding z to L would not have destroyed all out-branchings. Note that p_0 does not necessarily belong to T.

For the sake of simplifying the proof of Theorem 3.2.2 below we furthermore assume that the above algorithm picks the vertex x in line 8 in a depth-first manner. That is, the vertex x is chosen to be the last vertex added to T such that $x \in \text{Leaf}(T) - L$.

Theorem 3.2.1. *Algorithm $\mathcal{B}(D, T, L)$ works correctly. In other words, D has a (T, L)-out-branching with at least k leaves if and only if Algorithm $\mathcal{B}(D, T, L)$ returns “YES”.*

Proof. The only difference between $\mathcal{B}(D, T, L)$ and $\mathcal{A}(D, T, L)$ is that in line 12 we may add an extra vertex p_0 to L which was not done in $\mathcal{A}(D, T, L)$. We will now prove that this addition does not change the correctness of the algorithm.

So assume that there is an optimal (T, L)-out-branching T' with $x \in \text{Leaf}(T')$ but $p_0 \notin \text{Leaf}(T')$. We will show that this implies that an optimal solution is found in the

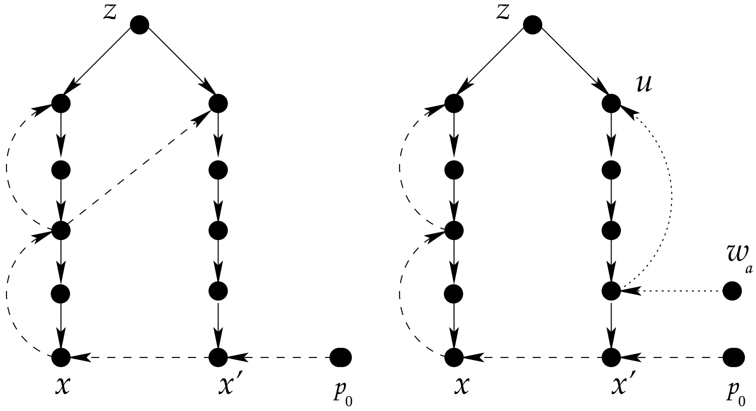

Figure 3.1: Real lines represents T'_z arcs; dashed lines represent the reachability of p_0; dotted lines represent the reachability of w_0.

branch of the search tree where we put z into L. This will complete the proof as if an optimal (T, L)-out-branching T' does not contain x as a leaf, by Lemma 3.1.5 it is found in $\mathcal{B}(D, T \cup T^{root}_{D,L}(x), L)$ and if it includes both x and p_0 as leaves then it is found in $B(D, T, L')$ (in line 15).

Note that $T_z = T'_z$ as T_z had exactly two leaves x and x' and $x' \in L$ and we have just assumed that x is a leaf of T'. Let $D^* = D[V(T'_z) \cup \{p_0\} - \{z\}]$ and consider the following two cases.

If p_0 can reach all vertices of D^* in D^* then proceed as follows. Let T^* be an out-branching in D^* with p_0 as the root. Let T'' be the out-branching obtained from T' by deleting all arcs in T'_z and adding all arcs in T^*. Note that $|\text{Leaf}(T'')| \geq |\text{Leaf}(T')|$ as $\text{Leaf}(T^*) \cup \{z\}$ are leaves in T'' and $\text{Leaf}(T'_z)$ are the only two leaves in T' which may not be leaves in T''. Therefore an optimal solution is found when we add z to L.

So now consider the case when p_0 cannot reach all vertices of D^* in D^*. This means that there is a vertex $u \in N^+_T(z)$ which cannot be reached by p_0 in D^*. All such unreachable vertices lie on the same branch of T_z (the branch not containing p_r). Let $W = w_0 w_1 w_2 \ldots w_l u$ be a path from the root of T to u, which does not use any arcs out of z (which exists as z was colored red in line 9, so adding z to L at this stage would not destroy all out-branchings). Assume that a is chosen such that $w_a \notin T'_z$ and $\{w_{a+1}, w_{a+2}, \ldots, w_l, u\} \subseteq V(T'_z)$ (see Figure 1).

Consider the digraph $D'' = D[V(T'_z) \cup \{p_0, w_a\} - \{z\}]$ and note that every vertex in D'' can be reached by either p_0 or w_a in D''. Therefore, there exists two vertex disjoint out-trees T_{p_0} and T_{w_a} rooted at p_0 and w_a, respectively, such that $V(T_{p_0}) \cup V(T_{w_a}) = V(D'')$ (to see that this claim holds add a new vertex y and two arcs yp_0 and yw_a). Furthermore since p_0 cannot reach u in D^* we note that both T_{p_0} and T_{w_a} must contain at least two vertices. Let T''' be the out-branching obtained from T' by deleting all arcs in T'_z and adding all arcs in T_{p_0} and in T_{w_a}. Note that $|\mathrm{Leaf}(T''')| \geq |\mathrm{Leaf}(T')|$ as $\mathrm{Leaf}(T_{p_0}) \cup \mathrm{Leaf}(T_{w_a}) \cup \{z\}$ are leaves in T''' and $\mathrm{Leaf}(T'_z) \cup \{w_a\}$ are the only three vertices which may be leaves in T' but not in T'''. Therefore again an optimal solution is found when we add z to L. □

Theorem 3.2.2. *Algorithm $\mathcal{B}(D, T, L)$ runs in time $O(3.72^k n^{O(1)})$.*

Proof. For an out-tree Q, let $\ell(Q) = |Leaf(Q)|$. Recall that we have assumed that $\mathcal{B}(D, T, L)$ picks the vertex x in line 8 in a depth-first manner.

Consider the search tree ST that we obtain by running the algorithm $\mathcal{B}(D, \{x\}, \emptyset)$. That is, the root of ST is the triple $(D, \{x\}, \emptyset)$. The children of this root is $(D, \{x\}, L')$ when we make a recursive call in line 15 and $(D, T^{root}_{D,L}(x), \emptyset)$ if we make a recursive call in line 16. The children of these nodes are again triples corresponding to the recursive calls.

Let $g(T, L)$ be the number of leaves in a subtree R of ST with triple (D, T, L). Clearly, $g(T, L) = 1$ when (D, T, L) is a leaf of ST. For a non-trivial subtree R of ST, we will prove, by induction, that $g(T, L) \leq c\alpha^{k-\ell(T)}\beta^{k-|L|}$, where $\alpha = 1.96$, $\beta = 1.896$ and $c \geq \alpha^2\beta^2$. Assume that this holds for all smaller non-trivial subtrees. (Note that the value of c is chosen in such a way that in the inequalities in the rest of the proof, we have upper bounds for $g(T^*, L^*)$ being at least 1 when (D, T^*, L^*) is a leaf of ST.)

Recall that $x \in \mathrm{Leaf}(T) - L$ was picked in line 8. Now consider the following possibilities.

If $|L'| = |L| + 2$, then the number of leaves of R is at most the following as if a call is made to $\mathcal{B}(D, T \cup T^{root}_{D,L}(x), L)$ in line 16 then the number of leaves of T increases by at least one:

$$\begin{aligned} g(T, L') + g(T \cup T^{root}_{D,L}(x), L) &\leq c\alpha^{k-\ell(T)}\beta^{k-|L|-2} + c\alpha^{k-\ell(T)-1}\beta^{k-|L|} \\ &= c\alpha^{k-\ell(T)}\beta^{k-|L|}\left(\tfrac{1}{\beta^2} + \tfrac{1}{\alpha}\right) \\ &\leq c\alpha^{k-\ell(T)}\beta^{k-|L|}. \end{aligned}$$

So we may assume that $|L'| = |L| + 1$ in line 15. Now assume that $|\mathrm{Leaf}(T^{root}_{D,L}(x))| \neq 2$ in line 16. In this case either no recursive call is made in line 16 or we increase the number of leaves in T by at least two. Therefore the number of leaves of R is at most

$$\begin{aligned} c\alpha^{k-\ell(T)}\beta^{k-|L|-1} + c\alpha^{k-\ell(T)-2}\beta^{k-|L|} &= c\alpha^{k-\ell(T)}\beta^{k-|L|}\left(\tfrac{1}{\beta} + \tfrac{1}{\alpha^2}\right) \\ &\leq c\alpha^{k-\ell(T)}\beta^{k-|L|}. \end{aligned}$$

So we may assume that $|L'| = |L| + 1$ in line 15 and $|\mathrm{Leaf}(T^{root}_{D,L}(x))| = 2$ in line 16. Let $T' = T \cup T^{root}_{D,L}(x)$ and consider the recursive call to $\mathcal{B}(D, T', L)$. If we increase the number of leaves in T' in lines 2-4 of the while-loop of this recursive call, then the number of leaves of the subtree of ST rooted at (D, T', L) is at most

$$c\alpha^{k-\ell(T')-1}\beta^{k-|L|-1} + c\alpha^{k-\ell(T')-2}\beta^{k-|L|} = c\alpha^{k-\ell(T')}\beta^{k-|L|}\left(\tfrac{1}{\alpha\beta} + \tfrac{1}{\alpha^2}\right).$$

Therefore, as $\ell(T') = \ell(T) + 1$, the number of leaves in R is at most

$$\begin{aligned} g(T,L') + g(T',L) &\leq c\alpha^{k-\ell(T)}\beta^{k-|L|-1} + c\alpha^{k-\ell(T)-1}\beta^{k-|L|}\left(\tfrac{1}{\alpha\beta} + \tfrac{1}{\alpha^2}\right) \\ &= c\alpha^{k-\ell(T)}\beta^{k-|L|}\left(\tfrac{1}{\beta} + \tfrac{1}{\alpha^2\beta} + \tfrac{1}{\alpha^3}\right) \\ &\leq c\alpha^{k-\ell(T)}\beta^{k-|L|}. \end{aligned}$$

So we may assume that we do not increase the number of leaves in lines 2-4 of the while-loop when we consider (D, T', L). Let y and y' denote the two leaves of T'_x (after possibly adding some arcs in the while-loop). Consider the recursive call to $\mathcal{B}(D, T', L \cup \{y\})$. If we increase the number of leaves of T' in the while-loop of lines 2-4 in this call then the number of leaves in R is at most

$$\begin{aligned} g(T, L\cup\{x\}) \ + \ g(T', L\cup\{y\}) \ + \ & g(T' \cup (T')^{root}_{D,L}(y), L) \\ \leq \ & c\alpha^{k-\ell(T)}\beta^{k-|L|}\left(\tfrac{1}{\beta} + (\tfrac{1}{\alpha^2\beta^2} + \tfrac{1}{\alpha^3\beta}) + \tfrac{1}{\alpha^2}\right) \\ \leq \ & c\alpha^{k-\ell(T)}\beta^{k-|L|}. \end{aligned}$$

So we may assume that we do not increase the number of leaves in lines 2-4 of the while-loop when we consider $(D, T', L \cup \{y\})$. However in this case we note that $|L'| = |L| + 2$ in this recursive call as when we consider y' the conditions of line 10 are satisfied as, in particular, T_x has exactly two leaves). So in this last case the number of leaves in R is at most

$$\begin{aligned} g(T, L\cup\{x\}) \ + \ g(T', L\cup\{y\}) \ + \ & g(T' \cup (T')^{root}_{D,L}(y), L) \\ \leq \ & c\alpha^{k-\ell(T)}\beta^{k-|L|}\left(\tfrac{1}{\beta} + (\tfrac{1}{\alpha\beta^3} + \tfrac{1}{\alpha^2\beta}) + \tfrac{1}{\alpha^2}\right) \\ \leq \ & c\alpha^{k-\ell(T)}\beta^{k-|L|}. \end{aligned}$$

We increase either $|L|$ or $\ell(T)$ whenever we consider a child in the search tree and no non-leaf in ST has $|L| \geq k$ or $\ell(T) \geq k$. Therefore, the number of nodes in ST is at most $O(k\alpha^k\beta^k) = O(3.72^k)$. As the amount of work we do in each recursive call is polynomial we get the desired time bound. □

3.3 Application: Exact Algorithm

Note that Directed Maximum Leaf can be solved in time $O(2^n n^{O(1)})$ by an exhaustive search using Lemma 2.1.1. Our $3.72^k n^{O(1)}$ algorithm for Directed k-Leaf yields an improvement for Directed Maximum Leaf, as follows.

Let $a = 0.526$. We can solve Directed Maximum Leaf for a digraph D on n vertices using the following algorithm ADML:

Stage 1. Set $k := \lceil an \rceil$. For each $x \in V(D)$ apply $\mathcal{B}(D, \{x\}, \emptyset)$ to decide whether D contains an out-branching with at least k leaves. If D contains such an out-branching, go to Stage 2. Otherwise, using binary search and $\mathcal{B}(D, \{x\}, \emptyset)$, return the maximum integer ℓ for which D contains an out-branching with ℓ leaves.

Stage 2. Set $\ell := \lceil an \rceil$. For $k = \ell + 1, \ell + 2, \ldots, n$, using Lemma 2.1.1, decide whether $\hat{D}(\emptyset, S)$ has an out-branching for any vertex set S of D of cardinality k and if the answer is "NO", return $k - 1$.

The correctness of ADML is obvious and we now evaluate its time complexity. Let $r = \lceil an \rceil$. Since $3.72^a < 1.996$, Stage 1 takes time at most $3.72^r n^{O(1)} = O(1.996^n)$. Since $\frac{1}{a^a(1-a)^{1-a}} < 1.9973$, Stage 2 takes time at most

$$\binom{n}{r} \cdot n^{O(1)} = \left(\frac{1}{a^a(1-a)^{1-a}} \right)^n n^{O(1)} = O(1.9973^n).$$

Thus, we obtain the following:

Theorem 3.3.1. *There is an algorithm to solve* Directed Maximum Leaf *in time* $O(1.9973^n)$.

3.4 Linear Kernel for Acyclic Digraphs

Lemma 2.1.1 implies that an acyclic digraph D has an out-branching if and only if D has a single vertex of in-degree zero. Since it is easy to check that D has a single vertex of in-degree zero, in what follows, we assume that the acyclic digraph D under consideration has a single vertex s of in-degree zero. In this section, a *k-out-branching* is short for an *out-branching* with at least k leaves.

We start from the following simple lemma.

Lemma 3.4.1. *In an acyclic digraph H with a single source s, every spanning subgraph of H, in which each vertex apart from s has in-degree 1, is an out-branching.*

Let B be an undirected bipartite graph with vertex bipartition (V', V''). A subset S of V' is called a *bidomination set* if for each $y \in V''$ there is an $x \in S$ such that $xy \in E(B)$. The so-called *greedy covering algorithm* [11] proceeds as follows: Start from the empty bidominating set C. While $V'' \neq \emptyset$ do the following: choose a vertex v of V' of maximum degree, add v to C, and delete v from V' and the neighbors of v from V''.

The following lemma have been obtained independently by several authors, see Proposition 10.1.1 in [11].

Lemma 3.4.2. *If the minimum degree of a vertex in V'' is d, then the greedy covering algorithm finds a bidominating set of size at most* $1 + \frac{|V_1|}{d}\left(1 + \ln \frac{d|V_2|}{|V_1|}\right)$.

Let D be an acyclic digraph with a single source. We use the following reduction rules to get rid of some vertices of in-degree 1.

(A) If D has an arc $a = xy$ with $d^+(x) = d^-(y) = 1$, then contract a.

(B) If D has an arc $a = xy$ with $d^+(x) \geq 2$, $d^-(y) = 1$ and $x \neq s$, then delete x and add arc uv for each $u \in N^-(x)$ and $v \in N^+(x)$.

The reduction rules are of interest due to the following:

Lemma 3.4.3. *Let D^* be the digraph obtained from an acyclic digraph D with a single source using Reduction Rules A and B as long as possible. Then D^* has a k-out-branching if and only if D has one.*

Proof. Let D have an arc $a = xy$ with $d^+(x) = d^-(y) = 1$ and let D' be the digraph obtained from D by contracting a. Let T be a k-out-branching of D. Clearly, T contains a and let T' be an out-branching obtained from T by contracting a. Observe that T' is also a k-out-branching whether y is a leaf of D or not. Similarly, if D' has a k-out-branching, then D has one, too.

Let D have an arc $a = xy$ with $d^+(x) \geq 2$, $d^-(y) = 1$ and $x \neq s$ and let D' be obtained from D by applying Rule B. We will prove that D' has a k-out-branching if and only if D has one. Let T be a k-out-branching in D. Clearly, T contains arc xy and x is not a leaf of T. Let U be the subset of $N^+(x)$ such that $xu \in A(T)$ for each $u \in U$ and let v be the vertex such that $vx \in A(T)$. Then the out-branching T' of D' obtained from T by deleting x and adding arcs vu for every $u \in U$ has at least k leaves (T' is an out-branching of D' by Lemma 3.4.1). Similarly, if D' has a k-out-branching, then D has one, too. □

Now consider D^*. Let B be an undirected bipartite graph, with vertex bipartition (V', V''), where V' is a copy of $V(D^*)$ and V'' is a copy of $V(D^*) - \{s\}$. We have $E(B) = \{u'v'' : u' \in V', v'' \in V'', uv \in A(D^*)\}$.

Lemma 3.4.4. *Let R be a bidominating set of B. Then D^* has an out-branching T such that the copies of the leaves of T in V' form a superset of $V' - R$.*

Proof. Consider a subgraph Q of B obtained from B by deleting all edges apart from one edge between every vertex in V'' and its neighbor in R. By Lemma 3.4.1, Q corresponds to an out-branching T of D^* such that the copies of the leaves of T in V' form a superset of $V' - R$. □

Theorem 3.4.5. *If D^* has no k-out-branching, then the number n^* of vertices in D^* is less than* $6.6(k + 2)$.

Proof. Suppose that $n^* \geq 6.6(k+2)$; we will prove that D^* has a k-out-branching. Observe that by Rules A and B, all vertices of D^* are of in-degree at least 2 apart from s and some of its out-neighbors. Let X denote the set of out-neighbors of s of in-degree 1 and let X'' be the set of copies of X in V''. Observe that the vertices of $V'' - X''$ of $B - X''$ are all of degree at least 2. Thus, by Lemma 3.4.2, $B - X''$ has a bidominating set S of size at most $\frac{n^*}{2}(1 + \ln 2) + 1$. Hence, $S \cup \{s\}$ is a bidominating set of B and, by Lemma 3.4.4, D^* has a b-out-branching with $b \geq n^* - \frac{n^*}{2}(1 + \ln 2) - 2$. It is not difficult to see that $b \geq \frac{n^*}{2}(1 - \ln 2) - 2 \geq 0.153n^* - 2 \geq k$ as $n^* \geq 6.6(k + 2)$. Therefore D^* has a k-out-branching. □

Chapter 4

k-Out-tree Problem

The k-Out-Tree problem is the problem of deciding for a given parameter k, whether an input digraph contains a given out-tree with $k \geq 2$ vertices. In this chapter, we present a randomized algorithms for k-Out-Tree of runtime $O^*(5.704^k)$ and a derandomized version of the algorithm whose running time is $O^*(6.14^k)$. The derandomized algorithm returns "YES" with good probability if the input digraph is a yes-instance while returning "NO" whenever it's a no-instance. We can make the error probability bounded by an arbitrarily small constant by executing the algorithm repeatedly.

Throughout this chapter we let $\mathrm{Leaf}(T)$ denotes the set of leaves in T for an out-tree T, and $\mathrm{Int}(T) = V(T) - \mathrm{Leaf}(T)$ stand for the set of internal vertices of T. Note that an input digraph D contains a copy of an out-tree T if there is an injection $f : V(T) \to V(D)$ such that $(f(u), f(v)) \in A(D)$ whenever $(u, v) \in A(T)$. Given such an injection f, or a copy of T in D equivalently, we say that the vertex $f(u)$ in $V(D)$ plays the role of $u \in V(T)$.

In Section 4.1, we describe and analyze the randomized algorithm for k-Out-Tree by Alon, Yuster and Zwick [8]. In Section 4.2, we present a new randomized algorithm for k-Out-Tree and and analyze its computational complexity. We derandomize our algorithm in Subsection 4.3.

4.1 Color-coding for k-Out-tree

Let $c : V(D) \to \{1, \ldots, k\}$ be a vertex k-coloring of a digraph D and let T be a k-vertex out-tree contained in D (as a subgraph). Then $V(T)$ and T are *colorful* if no pair of vertices of T are of the same color.

The following algorithm of [8] verifies whether D contains a colorful out-tree H such that H is isomorphic to T, when a coloring $c : V(D) \to \{1, \ldots, k\}$ is given. Note that a k-vertex subgraph H will be colorful with a probability of at least $k!/k^k > e^{-k}$. Thus, we can find a copy of T in D in e^k expected iterations of the following algorithm.

Algorithm 3 $\mathcal{L}(T, r)$

Require: A digraph D with a given coloring $c : V(D) \to \{1, \ldots, k\}$, an out-tree T on k vertices, a specified vertex r of T

Ensure: For each vertex u of D, $C_T(u) := \{C : C$ is a set of colors which appear on a colorful copy of T in D, where u plays the role of $r\}$.

```
 1: if $|V(T)| = 1$ then
 2:    for all $u \in V(D)$ do
 3:       Insert $\{c(u)\}$ into $C_T(u)$.
 4:    end for
 5:    Return $C_T(u)$ for each vertex $u$ of $D$.
 6: else
 7:    Choose an arc $(r', r'') \in A(T)$.
 8:    Let $T'$ and $T''$ be the subtrees of $T$ obtained by deleting $(r', r'')$, where $T'$ and $T''$
       contains $r'$ and $r''$, respectively.
 9:    Call $\mathcal{L}(T', r')$.
10:    Call $\mathcal{L}(T'', r'')$.
11:    for all $u \in V(D)$ do
12:       Compose the family of color sets $C_T(u)$ as follows:
13:       for all $(u, v) \in A(D)$ do
14:          for all $C' \in C_{T'}(u)$ and $C'' \in C_{T''}(v)$ do
15:             $C := C' \cup C''$ if $C' \cap C'' = \emptyset$
16:             Insert $C$ into $C_T(u)$.
17:          end for
18:       end for
19:    end for
20:    Return $C_T(u)$ for each vertex $u$ of $D$.
21: end if
```

Theorem 4.1.1. *Let T be an out-tree on k vertices and let $D = (V, A)$ be a digraph. A subgraph of D isomorphic to T, if one exists, can be found in $O(k(4e)^k \cdot |A|)$ expected time by running the algorithm $\mathcal{L}(T, r)$ for a random coloring c iteratively.*

Proof. Let $c : V(D) \to \{1, \ldots, k\}$ be a given coloring of D and suppose T' and T'' are the subtrees of T obtained in line 8. Let $|V(T')| = k'$ and $|V(T'')| = k''$, where $k' + k'' = k$. Then $|C_{T'}(u)| \le \binom{k-1}{k'-1}$ and $|C_{T'}(u)| \le \binom{k-1}{k''-1}$. Checking $C' \cap C'' = \emptyset$ takes $O(k)$ time. Hence, lines 11-19 require at most $\binom{k}{k/2}^2 \cdot k|A| \le k2^{2k}|A|$ operations.

Let $T(k)$ be the number of operations for $\mathcal{L}(T, r)$. We have the following recursion.

$$T(k) \le T(k') + T(k'') + k2^{2k-2}|A| \tag{4.1}$$

By induction, it is not difficult to check that $T(k) \le k4^k|A|$. Since the expected number of iterations of the algorithm $\mathcal{L}(T, r)$ is at most e^k, we achieve the claimed running time. □

Let C be a family of vertex k-colorings of a digraph D. We call C an (n, k)-*family of perfect hashing functions* if for each $X \subseteq V(D)$, $|X| = k$, there is a coloring $c \in C$ such that X is colorful with respect to c. One can derandomize the above algorithm of Alon et al. by using any (n, k)-family of perfect hashing functions in the obvious way. The time complexity of the derandomized algorithm depends of the size of the (n, k)-family of perfect hashing functions. Let $\tau(n, k)$ denote the minimum size of an (n, k)-family of perfect hashing functions. Nilli [95] proved that $\tau(n, k) \geq \Omega(e^k \log n/\sqrt{k})$. It is unclear whether there is an (n, k)-family of perfect hashing functions of size $O^*(e^k)$ [31], but even if it does exist, the running time of the derandomized algorithm would be $O^*((4e)^k)$.

4.2 Randomized FPT-algorithm for k-Out-tree

Before we introduce our new randomized algorithm for k-Out-Tree, we will give a brief account of the basic idea behind it. Let T be an out-tree on k vertices and let D be a digraph in which we want to find a copy of T. As in the randomized algorithm by Alon, Yuster and Zwick in [8], we break T into two subtrees T_w and T_b. However, unlike the former which deletes an arc of T, we break it by choosing a "splitting vertex" denoted as v^* and furthermore the resulting two subtrees overlap exactly on this splitting vertex v^*. Next we randomly partition the digraph D into two vertex-disjoint parts D_w and D_b, and then find a copy of T_w in D_w and a copy of T_b in D_b, if one exists. If we try sufficiently many partitions of D, it is possible to find a copy of T whenever D contains one as a subgraph (with some good probability in a randomized version of the algorithm, which can be derandomized consequently).

The trouble is that the fact D_w and D_b that contain copies of T_w and T_b, respectively does not necessarily means that D contains a copy of T as a whole. We need to ensure that there exist copies of T_w and T_b that actually overlap (and overlap only) on a vertex of D corresponding to the splitting vertex v^*. To this end, we allow some vertices of D_w, say S, to be shared by D_b by considering $D_b + S$ instead of D_b. Here S is the set of vertices in D_w that could correspond to the splitting vertex v^* of T_w. When we search for a copy of T_b in $D_b + S$, only those trees isomorphic to T_b in $D_b + S$ are considered legitimate where the vertex corresponding to v^* lies in S. In other words, we convey the information S obtained in the phase for T_w-D_w to the next phase for T_b-D_b so that we do not only ensure the global connectivity of $T_w + T_b = T$ in D but also reduce the search space for finding a copy of T_b in D_b.

Moreover, by conveying the information for v^* we can save the extra effort for "merging" the solutions (i.e. copies of T_w and T_b). Rather, once we obtain a copy of T_b in $D_b + S$, it follows immediately that we have a copy of T in D. Since the number of partitions of D

we need to try is a function of k, the time complexity of finding a copy of T in D can be written as $T(k, n) = f(k)(T(k', n)+T(k-k', n)+p_1(n))+p_2(n)$, $T(1, n) = p_3(n)$, where $p_i(n)$ is polynomial in n for $i = 1, 2, 3$. This is why the running time of our algorithm remains polynomial in n. Making this approach efficient depends crucially on two aspects:

1. to obtain k' in the above formula as close to half of k as possible; and
2. to replace $f(k)$ with as small growing function as possible.

For the latter, we use a simple *unbalanced-partition-strategy* which will be explained later. We achieve the former goal by choosing an appropriate splitting vertex v^* and then using it to obtain a T_w-T_b split. The splitting procedure is one of the key parts of our algorithm and next we describe this procedure in details.

The following lemma is well known and will be used as a basic scheme of choosing v^*.

Lemma 4.2.1 ([34]). *Let T be an undirected tree and let $w : V \to \mathbb{R}^+ \cup \{0\}$ be a weight function on its vertices. There exists a vertex $v \in V(T)$ such that the weight of every subtree T' of $T - v$ is at most $w(T)/2$, where $w(T) = \sum_{v \in V(T)} w(v)$.*

Consider a partition $n = n_1 + \cdots + n_q$, where n and all n_i are nonnegative integers and a bipartition (A, B) of the set $\{1, \ldots, q\}$. Let $d(A, B) := \left| \sum_{i \in A} n_i - \sum_{i \in B} n_i \right|$. Given a set $Q = \{1, \ldots, q\}$ with a nonnegative integer weight n_i for each element $i \in Q$, we say that a bipartition (A, B) of Q is *greedily optimal* if $d(A, B)$ does not decrease by moving an element of one partite set into another. The following procedure describes how to obtain a greedily optimal bipartition in time $O(q \log q)$. For simplicity we write $\sum_{i \in A} n_i$ as $n(A)$.

Algorithm 4 Bipartition($Q, \{n_i : i \in Q\}$)

```
Require: A set $Q = \{1, \ldots, q\}$ with a nonnegative integer weight $n_i$, $\forall i \in Q$
Ensure: A greedily optimal bipartition $(A, B)$ of $Q$
Let $A := \emptyset$, $B := Q$.
while $n(A) < n(B)$ and there is an element $i \in B$ with $0 < n_i < d(A, B)$ do
    Choose such an element $i \in B$ with a largest $n_i$ with $n_i < d(A, B)$.
    $A := A \cup \{i\}$ and $B := B - \{i\}$.
end while
Return $(A, B)$.
```

Lemma 4.2.2. *Let Q be a set of size q with a nonnegative integer weight n_i for each $i \in Q$. The algorithm* **Bipartition**($Q, \{n_i : i \in Q\}$) *finds a greedily optimal bipartition $A \cup B = Q$ in time $O(q \log q)$.*

Proof. First we want to show that the values n_i chosen in line 3 of the algorithm do not increase during the performance of the algorithm. The values of n_i do not increase because the values of the difference $d(A, B)$ do not increase during the performance of the algorithm. In fact, $d(A, B)$ strictly decreases. To see this, suppose that the element i is selected in the present step. If $n(A \cup \{i\}) < n(B - \{i\})$, then obviously the difference $d(A, B)$ strictly decreases. Else if $n(A \cup \{i\}) > n(B - \{i\})$, we have $d(A \cup \{i\}, B - \{i\}) < n_i < d(A, B)$.

To see that the algorithm returns a greedily optimal bipartition (A, B), it is enough to observe that for the final bipartition (A, B), moving any element of A or B does not decrease $d(A, B)$. Suppose that the last movement of the element i_0 makes $n(A) > n(B)$. Then a simple computation implies that $d(A, B) < n_{i_0}$. Since the values of n_i in line 3 of the algorithm do not increase during the performance of the algorithm, $n_j \geq n_{i_0} > d(A, B)$ for every $j \in A$, the movement of any element in A would not decrease $d(A, B)$. On the other hand suppose that $n(A) < n(B)$. By the definition of the algorithm, for every $j \in B$ with a positive weight we have $n_j \geq d(A, B)$ and thus the movement of any element in B would not decrease $d(A, B)$. Hence the current bipartition (A, B) is greedily optimal.

Now let us consider the running time of the algorithm. Sorting the elements in non-decreasing order of their weights will take $O(q \log q)$ time. Moreover, once an element is moved from one partite set to another, it will not be moved again and we move at most q elements without duplication during the algorithm. This gives us the running time of $O(q \log q)$. □

Now we describe a new randomized algorithm for k-Out-Tree. Let D be a digraph and let T be an out-tree on k vertices. Let us specify a vertex $t \in V(T)$ and a vertex $w \in V(D)$. We call a copy of T in D a T-*isomorphic* tree. We say that a T-isomorphic tree T_D in D is a (t, w)-tree if $w \in V(T_D)$ plays the role of t.

We first give an intuitive explanation of our algorithm before giving a formal description. To find the desired tree in the given input digraph, we first split the tree in two parts with one common vertex such that the both parts are "almost balanced" Then we randomly partition the vertices of the D in two parts with probability of a vertex lying in one part or the other depends on the sizes of the trees we obtained in the first step by splitting it on a vertex. This allows us to more or less independently look for the different parts of the tree in different parts of the partition. We finally merge them cleverly to obtain our solution.

In the following algorithm **find-tree**, we have several arguments other than the natural arguments T and D. Our next argument is a vertex t of T. The argument t indicates that we want to return, at the end of the current procedure, the set of vertices X_t such that there is a (t, w)-tree for every $w \in X_t$. The fact that $X_t \neq \emptyset$ means two points: we have a T-isomorphic tree in D, and the information X_t we have can be used to construct a larger

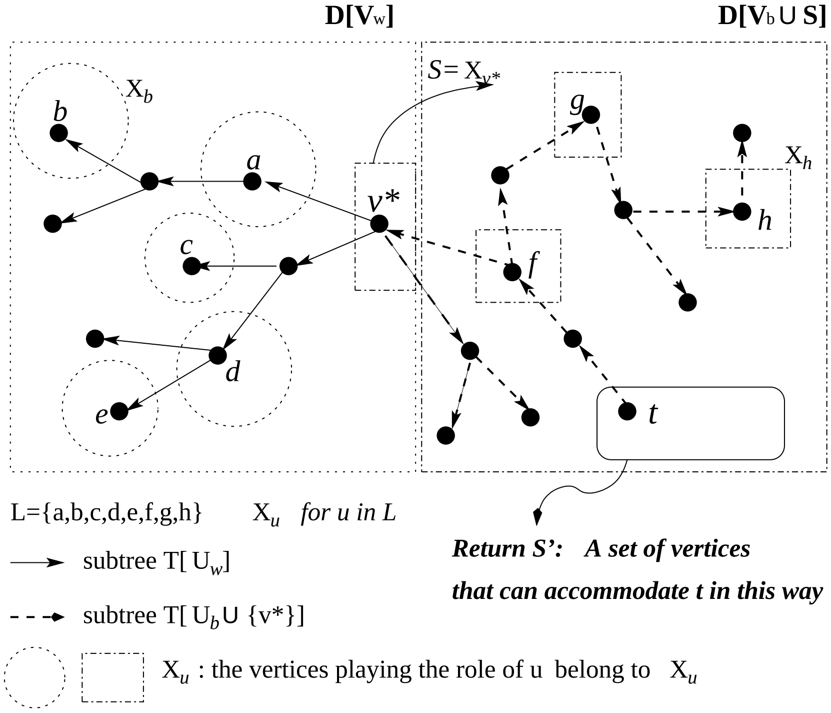
D[Vw]
D[Vb U S]
S= X_v*
X_b
b
a
c
d
e
v*
g
f
h
X_h
t
L={a,b,c,d,e,f,g,h}
X_u for u in L
subtree T[U_w]
subtree T[U_b U {v*}]
X_u : the vertices playing the role of u belong to X_u
Return S': A set of vertices
that can accommodate t in this way

Figure 4.1: An example: The given out-tree T is divided into two parts $T[U_w]$ and $T[U_b \cup \{v^*\}]$ by the splitting vertex v^*. The digraph D contains a copy of T meeting the restrictions on L.

tree which uses the current T-isomorphic tree as a building block. Here, X_t is a kind of 'joint'.

The basic strategy is as follows. We choose a pair T_A and T_B of subtrees of T such that $V(T_A) \cup V(T_B) = V(T)$ and T_A and T_B share only one vertex, namely v^*, the splitting vertex. We call recursively two '**find-tree**' procedures on subsets of $V(D)$ to ensure that the subtrees playing the role of T_A and T_B do not overlap. The first call (line 15) tries to find X_{v^*} and the second one (line 18), using the information X_{v^*} delivered by the first call, tries to find X_t.

We also need another argument to our algorithm **find-tree** which is useful while merging and that is:

- a pair consisting of $L \subseteq V(T)$ and $\{X_u : u \in L\}$, where $X_u \subset V(D)$ and X_u's are pairwise disjoint.

The arguments $L \subseteq V(T)$ and $\{X_u : u \in L\}$ form a set of information needed to argue the correctness of the algorithm. Essentially L is a set of vertices of the tree T which has been used as a splitting vertex at some point during the execution of our recursive procedure. Let T_D be a T-isomorphic tree; if for every $u \in L$, T_D is a (u, w)-tree for some $w \in X_u$ and $V(T_D) \cap X_u = \{w\}$, we say that T_D *meets the restrictions on L*. The algorithm **find-tree** intends to find the set X_t of vertices such that for every $w \in X_t$, there is a (t, w)-tree which meets the restrictions on L.

Deleting a splitting vertex v^* may produce several subtrees, and there might be many ways to divide them into two groups, namely $(T_A,\ T_B)$. To make the algorithm more efficient, we try to obtain as 'balanced' a partition $(T_A,\ T_B)$ as possible. The algorithm **tree-Bipartition** is used to produce a pretty 'balanced' bipartition of the subtrees. Moreover we introduce another argument to have a better complexity behavior. The argument v is a vertex which indicates whether there is a predetermined splitting vertex. If $v = \emptyset$, we do not have a predetermined splitting vertex so we find one in the current procedure. Otherwise, we use the vertex v as a splitting vertex.

Let r be the root of T. To decide whether D contains a copy of T, it suffices to run **find-tree**$(T, D, \emptyset, r, \emptyset, \emptyset)$.

Lemma 4.2.3. *During the performance of find-tree$(T, D, \emptyset, r, \emptyset, \emptyset)$, the sets X_u, $u \in L$ are pairwise disjoint.*

Proof. We prove the claim inductively. For the initial call, trivially the sets X_u, $u \in L$ are pairwise disjoint since $L = \emptyset$. Suppose that for a call find-tree$(T, D, v, t, L, \{X_u :\ u \in L\})$ the sets X_v, $v \in L$ are pairwise disjoint. For the first subsequent call in line 15, the sets are obviously pairwise disjoint. Consider the second subsequent call in line 18. If $v^* \in L$ before line 17, the claim is true since we convey the argument $t := v^*$ to the first subsequent call in line 15 and thus S is contained in X_{v^*}. Otherwise, observe that $X_u \subseteq V_b$ for all $u \in L \cap U_b$ and they are pairwise disjoint. Since $X_{v^*} \cap V_b = \emptyset$, the sets X_u for all $u \in L \cap U_b$ together with X_{v^*} are pairwise disjoint. □

The algorithm **tree-Bipartition** is a subroutine used during the execution of **find-tree**. Let $T_1, \dots, T_q$ be the subtrees of $T - v^*$, where v^* is a splitting vertex of the current call to **find-tree**. At the end of **tree-Bipartition**, we obtain a partition of the subtrees, or more precisely, a partition (WH, BL) of the indices $\{1, \dots, q\}$ of the subtrees. The attained partition (WH, BL) is 'a greedily optimal bipartition' in certain sense while a nonnegative integer weight on an element of $\{1, \dots, q\}$ is set to be $w(T_i)$ with some fine-tuning.

Lemma 4.2.4. *Consider the algorithm* **tree-Bipartition** *and let (WH, BL) be a bipartition of $\{1, \dots, q\}$ obtained at the end of the algorithm. Then the partition $U_w := \bigcup_{i \in WH} V(T_i) \cup \{v^*\}$ and $U_b := \bigcup_{i \in BL} V(T_i)$ of $V(T)$ has the the following property.*
1) If $v^* = t$, moving a component T_i from one partite set to the other does not decrease the difference $d(w(U_w), w(U_b))$.
2) If $v^* \neq t$, either exchanging v^* and the component T_l or moving a component T_i, $i \neq v^*, l$ from one partite set to the other does not decrease the difference $d(w(U_w), w(U_b))$.

```
Algorithm 5 find-tree($T, D, v, t, L, \{X_u : u \in L\}$)
Require: An out-tree $T$ on $k$ vertices, a digraph $D$, $v \in \{\emptyset\} \cup V(T)$, a specified vertex $t \in V(T)$, a subset of vertices $L \subseteq V(T)$, a family of pairwise disjoint subsets $X_u \subseteq V(D)$ for each $u \in L$.
Ensure: A set of vertices $X_t \subseteq V(D)$ such that there is a $(t, w)$-tree which meets the restriction on $L$ for every $w \in X_t$.
if $|V(T) \setminus L| \geq 2$ then
    for all $u \in V(T)$: Set $w(u) := 0$ if $u \in L$, $w(u) := 1$ otherwise.
    if $v = \emptyset$ then Find $v^* \in V(T)$ such that the weight of every subtree $T'$ of $T - v^*$ is at most $w(T)/2$ (see Lemma 4.2.1) else $v^* := v$
    $(WH, BL)$:=tree-Bipartition($T, t, v^*, L$).
    $U_w := \bigcup_{i \in WH} V(T_i) \cup \{v^*\}$, $U_b := \bigcup_{i \in BL} V(T_i)$.
    for all $u \in L \cap U_w$: color all vertices of $X_u$ in white.
    for all $u \in L \cap (U_b \setminus \{v^*\})$: color all vertices of $X_u$ in black.
    $\alpha := \min\{w(U_w)/w(T), w(U_b)/w(T)\}$.
    if $\alpha^2 - 3\alpha + 1 \leq 0$ (i.e., $\alpha \geq (3 - \sqrt{5})/2$, see (4.2) and the definition of $\alpha^*$ afterwards) then $v_w := v_b := \emptyset$
    else if $w(U_w) < w(U_b)$ then $v_w := \emptyset$, $v_b := v^*$ else $v_w := v^*$, $v_b := \emptyset$.
    $X_t := \emptyset$.
    for $i = 1$ to $\left\lceil \frac{2.51}{\alpha^{\alpha k}(1-\alpha)^{(1-\alpha)k}} \right\rceil$ do
        Color the vertices of $V(D) - \bigcup_{u \in L} X_u$ in white or black such that for each vertex the probability to be colored in white is $\alpha$ if $w(U_w) \leq w(U_b)$, and $1 - \alpha$ otherwise.
        Let $V_w$ ($V_b$) be the set of vertices of $D$ colored in white (black).
        $S$ :=find-tree($T[U_w], D[V_w], v_w, v^*, L \cap U_w, \{X_u : u \in L \cap U_w\}$)
        if $S \neq \emptyset$ then
            $X_{v^*} := S$, $L := L \cup \{v^*\}$.
            $S'$ :=find-tree($T[U_b \cup \{v^*\}], D[V_b \cup S], v_b, t, (L \cap U_b), \{X_u : u \in (L \cap U_b)\}$).
            $X_t := X_t \cup S'$.
        end if
    end for
    Return $X_t$.
else $\{|V(T) \setminus L| \leq 1\}$
    if $\{z\} = V(T) \setminus L$ then $X_z := V(D) - \bigcup_{u \in L} X_u$, $L := L \cup \{z\}$.
    $L^o := \{$all leaf vertices of $T\}$.
    while $L^o \neq L$ do
        Choose a vertex $z \in L \setminus L^o$ s.t. $N_T^+(z) \subseteq L^o$.
        $X_z := X_z \cap \bigcap_{u \in N_T^+(z)} N^-(X_u)$; $L^o := L^o \cup \{z\}$.
    end while
    Return $X_t$.
end if
```

Proof. Let us consider the property 1). The bipartition (WH, BL) is determined in the first 'if' statement in line 2 of **tree-Bipartition**. Then by Lemma 4.2.2 the bipartition (WH, BL) is greedily optimal, which is equivalent to the statement of 1).

Algorithm 6 tree-Bipartition(T, t, v^*, L)

```
 1: $T_1, \dots, T_q$ are the subtrees of $T - v^*$. $Q := \{1, \dots, q\}$. $w(T_i) := |V(T_i) \setminus L|, \forall i \in Q$.
 2: if $v^* = t$ then
 3:    $(A, B)$:=Bipartition$(Q, \{n_i := w(T_i) : i \in Q\})$
 4:    if $w(A) \le w(B)$ then $WH := A$, $BL := B$. else $WH := B$, $BL := A$.
 5: else
 6:    Let $l$ be such that $t \in V(T_l)$
 7:    if $w(T_l) - w(v^*) \ge 0$ then
 8:       $(A, B)$:=Bipartition$(Q, \{n_i := w(T_i) : i \in Q \setminus \{l\}\} \cup \{n_l := w(T_l) - w(v^*)\})$
 9:       if $l \in B$ then $WH := A$, $BL := B$. else $WH := B$, $BL := A$
10:    else $\{w(T_l) - w(v^*) < 0\}$
11:       $(A, B)$:=Bipartition$((Q \setminus \{l\}) \cup \{v^*\}, \{n_i := w(T_i) : i \in Q \setminus \{l\}\} \cup \{n_{v^*} := w(v^*)\})$
12:       if $v^* \in A$ then $WH := A - \{v^*\}$, $BL := B \cup \{l\}$. else $WH := B - \{v^*\}$, $BL := A \cup \{l\}$
13:    end if
14: end if
15: Return $(WH, BL)$.
```

Let us consider the property 2). First suppose that the bipartition (WH, BL) is determined in the ‘if’ statement in line 7 of **tree-Bipartition**. The exchange of v^* and the component T_l amounts to moving the element l in the algorithm **Bipartition**. Since (WH, BL) is returned by **Bipartition** and thus is a greedily optimal bipartition of Q, any move of an element in one partite set would not decrease the difference $d(WH, BL)$ and the statement of 2) holds in this case.

Secondly suppose that the bipartition (WH, BL) is determined in the ‘if’ statement in line 10 of **tree-Bipartition**. In this case we have $w(T_l) = 0$ and thus exchanging T_l and v^* and amounts to moving the element v^* in the algorithm **Bipartition**. By the same argument as above, any move of an element in one partite set would not decrease the difference $d(WH, BL)$ and again the statement of 2) holds. □

Consider the following equation:

$$\alpha^2 - 3\alpha + 1 = 0 \tag{4.2}$$

Let $\alpha^* := (3 - \sqrt{5})/2$ be one of its roots. In line 10 of the algorithm **find-tree**, if $\alpha < \alpha^*$ we decide to pass the present splitting vertex v^* as a splitting vertex to the next recursive call which gets, as an argument, a subtree with greater weight among the two subtrees $T[U_w]$ and $T[U_b \cup \{v^*\}]$. Lemma 4.2.5 justifies this execution. It claims that if $\alpha < \alpha^*$, then in the next recursive call with a subtree of weight $(1 - \alpha)w(T)$, we have a more balanced bipartition with v^* as a splitting vertex. Actually, the bipartition in the next step is good enough so as to compensate for the increase in the running time incurred by the biased (‘$\alpha < \alpha^*$’) bipartition in the present step. We will show this later.

Lemma 4.2.5. *Suppose that v^* has been chosen to split T for the present call to* **find-tree** *such that the weight of every subtree of $T - v^*$ is at most $w(T)/2$ and that $w(T) \geq 5$. Let α be defined as in line* 8 *and assume that $\alpha < \alpha^*$. Let $\{U_1, U_2\} = \{U_w, U_b\}$ such that $w(U_2) \geq w(U_1)$ and let $\{T_1, T_2\} = \{T[U_w], T[U_b \cup \{v^*\}]\}$ such that $U_1 \subseteq V(T_1)$ and $U_2 \subseteq V(T_2)$. Let α' play the role of α in the recursive call using the tree T_2. In this case the following holds: $\alpha' \geq (1 - 2\alpha)/(1 - \alpha) > \alpha^*$.*

Proof. Let $T_1, T_2, U_1, U_2, \alpha, \alpha'$ be defined as in the statement. Note that $\alpha = w(U_1)/w(T)$. Let $d = w(U_2) - w(U_1)$ and note that $w(U_1) = (w(T) - d)/2$ and that the following holds

$$\frac{1 - 2\alpha}{1 - \alpha} = \frac{w(T) - 2w(U_1)}{w(T) - w(U_1)} = \frac{2d}{w(T) + d}.$$

We now consider the following cases.

Case 1. $d = 0$: In this case $\alpha = 1/2 > \alpha^*$, a contradiction.

Case 2. $d = 1$: In this case $\alpha^* > \alpha = w(U_1)/(2w(U_1) + 1)$, which implies that $w(U_1) \leq 1$. Therefore $w(U_2) \leq 2$ and $w(T) \leq 3$, a contradiction.

Case 3. $d \geq 2$: Let $C_1, C_2, \ldots, C_q$ denote the components in $T - v^*$ and without loss of generality assume that $V(C_1) \cup V(C_2) \cup \cdots \cup V(C_a) = U_2$ and $V(C_{a+1}) \cup V(C_{a+2}) \cup \cdots \cup V(C_q) = U_1$. Note that by Lemma 4.2.4 we must have $w(C_i) \geq d$ or $w(C_i) = 0$ for all $i = 1, 2, \ldots, q$ except possibly for one set C_l (containing t), which may have $w(C_l) = 1$ (if $w(v^*) = 1$).

Let C_r be chosen such that $w(C_r) \geq d$, $1 \leq r \leq a$ and $w(C_r)$ is minimum possible with these constraints. We first consider the case when $w(C_r) > w(U_2) - w(C_r)$. By the above (and the minimality of $V(C_r)$) we note that $w(U_2) \leq w(C_r) + 1$ (as either C_j, which is defined above, or v^* may belong to $V(T_2)$, but not both). As $w(U_2) = (w(T) + d)/2 \geq w(T)/2 + 1$ we note that $w(C_r) \geq w(T)/2 + d/2 - 1$. As $w(C_r) \leq w(T)/2$ (By the statement in our theorem) this implies that $d = 2$ and $w(C_r) = w(T)/2$ and $w(U_2) = w(C_r) + 1$. If U_1 contains at least two distinct components with weight at least d then $w(U_1) > w(U_2)$, a contradiction. If U_1 contains no component of weight at least d then $w(U_1) \leq 1$ and $w(T) \leq 4$, a contradiction. So U_1 contains exactly one component of weight at least d. By the minimality of $w(C_r)$ we note that $w(U_1) \geq w(C_r) = w(U_2) - 1$, a contradiction to $d \geq 2$.

Therefore we can assume that $w(C_r) \leq w(U_2) - w(C_r)$, which implies the following (the last equality is proved above)

$$\alpha' \geq \frac{w(C_r)}{w(U_2)} \geq \frac{d}{(w(T) + d)/2} = \frac{1 - 2\alpha}{1 - \alpha}.$$

As $\alpha < \alpha^*$, we note that $\alpha' \geq (1 - 2\alpha)/(1 - \alpha) > (1 - 2\alpha^*)/(1 - \alpha^*) = \alpha^*$. □

For the selection of the splitting vertex v^* we have two criteria in the algorithm **find-tree**: (i) *'found'* criterion: the vertex is found so that the weight of every subtree T' of $T - v^*$ is at most $w(T)/2$. (ii) *'taken-over'* criterion: the vertex is passed on to the present step as the argument v by the previous step of the algorithm. The following statement is an easy consequence of Lemma 4.2.5.

Corollary 4.2.6. *Suppose that $w(T) \geq 5$. If v^* is selected with 'taken-over' criterion, then $\alpha > \alpha^*$.*

Proof. For the initial call find-tree$(T, D, \emptyset, r, \emptyset, \emptyset)$ we have $v = \emptyset$ and thus, the splitting vertex v^* is selected with the 'found' criterion. We will prove the claim by induction. Consider the first vertex v^* selected with then 'taken-over' criterion during the performance of the algorithm. Then in the previous step, the splitting vertex was selected with 'found' criterion and thus in the present step we have $\alpha > \alpha^*$ by Lemma 4.2.5.

Now consider a vertex v^* selected with the 'taken-over' criterion. Then in the previous step, the splitting vertex was selected with the 'found' criterion since otherwise, by the induction hypothesis we have $\alpha > \alpha^*$ in the previous step, and $\emptyset$ has been passed on as the argument v for the present step. This is a contradiction. □

Due to Corollary 4.2.6 the vertex v^* selected in line 3 of the algorithm **find-tree** functions properly as a splitting vertex. In other words, we have more than one subtree of $T - v^*$ in line 4 with positive weights.

Lemma 4.2.7. *If $w(T) \geq 2$, then for each of U_w and U_b found in line 5 of by* **find-tree** *we have $w(U_w) > 0$ and $w(U_b) > 0$.*

Proof. For the sake of contradiction suppose that one of $w(U_w)$ and $w(U_b)$ is zero. Let us assume $w(U_w) = 0$ and $w(U_b) = w(T)$. If v^* is selected with 'found' criteria, each component in $T[U_b]$ has a weight at most $w(T)/2$ and $T[U_b]$ contains at least two components of positive weights. Then we can move one component with a positive weight from U_b to U_w which will reduce the difference $d(U_w, U_b)$, a contradiction. The same argument applies when $w(U_w) = w(T)$ and $w(U_b) = 0$.

Consider the case when v^* is selected with "taken-over" criteria. There are three possibilities.

Case 1. $w(T) \geq 5$*:* In this case we obtain a contradiction with Corollary 4.2.6.

Case 2. $w(T) = 4$*:* In the previous step using T', where $T \subseteq T'$, the splitting vertex v^* was selected with "found" criteria. Then by the argument in the first paragraph, we have $w(T') \geq 5$. A contradiction follows from Lemma 4.2.5.

Case 3. $2 \leq w(T) \leq 3$*:* First suppose that $w(v^*) = 0$. Note that $T[U_w] - v^*$ or $T[U_b]$ contains a component of weight $w(T)$ since otherwise we can move a component with a positive weight from one partite set to the other and reduce $d(U_w, U_b)$. Considering the previous step using T', where $T \subseteq T'$, the out-tree T is the larger of T'_w and T'_b. We pass the splitting vertex v^* to the larger of the two only when $\alpha < \alpha^*$. So when $w(T) = 3$, we have $3 > (1-\alpha^*)w(T')$ and thus $w(T') \leq 4$, and when $w(T) = 2$ we have $2 > (1-\alpha^*)w(T')$ and thus $w(T') \leq 3$. In either case, however, $T' - v^*$ contains a component with a weight greater than $w(T')/2$, contradicting to the choice of v^* in the previous step (Recall that v^* is selected with 'found' criteria in the previous step using T').

Secondly suppose that that $w(v^*) = 1$. Then $w(U_w) = w(T)$ and $w(U_b) = 0$. We can reduce the difference $d(U_w, U_b)$ by moving the component with a positive weight from U_w to U_b, a contradiction.

Therefore for each of U_w and U_b found in line 5 of by **find-tree** we have $w(U_w) > 0$ and $w(U_b) > 0$. □

Lemma 4.2.8. *Given a digraph D, an out-tree T and a specified vertex $t \in V(T)$, consider the set X_t (in line 22) returned by the algorithm* **find-tree***$(T, D, v, t, L, \{X_u : u \in L\})$. We assume that the sets X_u, $u \in L$ are pairwise disjoint. If $w \in X_t$ then D contains a (t, w)-tree that meets the restrictions on L. Conversely, if D contains a (t, w)-tree for a vertex $w \in V(D)$ that meets the restrictions on L, then X_t contains w with probability larger than $1 - 1/e > 0.6321$.*

Proof. Lemma 4.2.7 guarantees that the splitting vertex v^* selected at any recursive call of **find-tree** really 'splits' the input out-tree T into two nontrivial parts, unless $w(T) \leq 1$.

First we show that if $w \in X_t$ then D contains a (t, w)-tree for a vertex $w \in V(D)$ that meets the restrictions on L. When $|V(T) \setminus L| \leq 1$, using Lemma 4.2.3 it is straightforward to check from the algorithm that the claim holds. Assume that the claim is true for all subsequent calls to **find-tree**. Since $w \in S'$ for some S' returned by a call in line 18, the subgraph $D[V_b \cup X_{v^*}]$ contains a $T[U_b \cup \{v^*\}]$-isomorphic (t, w)-tree T_D^b meeting the restrictions on $(L \cap U_b) \cup \{v^*\}$ by induction hypothesis. Moreover, $X_{v^*} \neq \emptyset$ when $S' \ni w$ is returned and this implies that there is a vertex $u \in X_{v^*}$ such that T_D^b is a (v^*, u)-tree. Since $u \in X_{v^*}$, induction hypothesis implies that the subgraph $D[V_w]$ contains a $T[U_w]$-isomorphic (v^*, u)-tree, say T_D^w.

Consider the subgraph $T_D := T_D^w \cup T_D^b$. To show that T_D is a T-isomorphic (t, w)-tree in D, it suffices to show that $V(T_D^w) \cap V(T_D^b) = \{u\}$. Indeed, $V(T_D^w) \subseteq V_w$, $V(T_D^b) \subseteq V_b \cup X_{v^*}$ and $V_w \cap V_b = \emptyset$. Thus if two trees T_D^w and T_D^b share vertices other than u, these common vertices should belong to X_{v^*}. Since T_D^b meets the restrictions on $(L \cap U_b) \cup \{v^*\}$, we have $X_{v^*} \cap V(T_D^b) = \{u\}$. Hence u is the only vertex that two trees T_D^w and T_D^b have in common. We know that u plays the role of v^* in both trees. Therefore we conclude that T_D is T-isomorphic, and since w plays the role of t, it is a (t, w)-tree. Obviously T_D meets the restrictions on L.

Secondly, we shall show that if D contains a (t, w)-tree for a vertex $w \in V(D)$ that meets the restrictions on L, then X_t contains w with probability larger than $1 - 1/e > 0.6321$. When $|V(T) \setminus L| \leq 1$, the algorithm **find-tree** is deterministic and returns X_t which is exactly the set of all vertices w for which there exists a (t, w)-tree meeting the restrictions on L. Hence the claim holds for the base case, and we may assume that the claim is true for all subsequent calls to **find-tree**.

Suppose that there is a (t, w)-tree T_D meeting the restrictions on L and that this is a (v^*, w')-tree, that is, the vertex w' plays the role of v^*. Then the vertices of T_D corresponding to U_w, say T_D^w, are colored white and those of T_D corresponding to U_b, say T_D^b, are colored black as intended with probability $\geq (\alpha^\alpha(1 - \alpha)^{1-\alpha})^k$. When we hit the right coloring for T, the digraph $D[V_w]$ contains the subtree T_D^w of T_D which is $T[U_w]$-isomorphic and which is a (v^*, w')-tree. By induction hypothesis, the set S obtained in line 15 contains w' with probability larger than $1 - 1/e$. Note that T_D^w meets the restrictions on $L \cap U_w$.

If $w' \in S$, the restrictions delivered onto the subsequent call for **find-tree** in line 17 contains w'. Since T_D meets the restrictions on L confined to $U_b - v^*$ and it is a (v^*, w')-tree with $w' \in S = X_{v^*}$, the subtree T_D^b of T_D which is $T[U_b \cup \{v^*\}]$-isomorphic meets all the restrictions on L. Hence by induction hypothesis, the set S' returned in line 18 contains w with probability larger than $1 - 1/e$.

The probability ρ that S', returned by **find-tree** in line 18 at an iteration of the loop, contains w is, thus,

$$\rho > (\alpha^\alpha(1 - \alpha)^{1-\alpha})^k \times (1 - 1/e)^2 > 0.3995(\alpha^\alpha(1 - \alpha)^{1-\alpha})^k.$$

After looping $\lceil(0.3995(\alpha^\alpha(1 - \alpha)^{1-\alpha})^k)^{-1}\rceil$ times in line 12, the probability that X_t contains w is at least

$$1 - (1 - \rho)^{1/(0.3995(\alpha^\alpha(1-\alpha)^{1-\alpha})^k)} > 1 - (1 - 0.3995(\alpha^\alpha(1 - \alpha)^{1-\alpha})^k)^{1/(0.3995(\alpha^\alpha(1-\alpha)^{1-\alpha})^k)} > 1 - \frac{1}{e}.$$

Observe that the probability ρ does not depend on α and the probability of coloring a vertex white/black. □

4.2.1 Running Time Analysis

The complexity of Algorithm **find-tree** is analyzed in the following theorem.

Theorem 4.2.9. *Algorithm* **find-tree** *has running time $O(n^2 k^\rho C^k)$, where $w(T) = k$ and*

$|V(D)| = n$, and C and ρ are defined and bounded as follows:

$$C = \left(\frac{1}{\alpha^{*\alpha^*}(1-\alpha^*)^{1-\alpha^*}} \right)^{1/\alpha^*},\ \rho = \frac{\ln(1/6)}{\ln(1-\alpha^*)},\ \rho \leq 3.724,\ \textit{and}\ C \leq 5.7039.$$

Proof. Let $L(T, D)$ denote the number of times the 'if'-statement in line 1 of Algorithm **find-tree** is false (in all recursive calls to **find-tree**). We will prove that $L(T, D) \leq R(k) = Bk^{\rho}C^k + 1$, $B \geq 1$ is a constant whose value will determined later in the proof. This would imply that the number of calls to **find-tree** where the 'if'-statement in line 1 is true is also bounded by $R(k)$ as if line 1 is true then we will have at least two calls to **find-tree** (in fact it will have at least three as $\left\lceil \frac{2.51}{\alpha^{\alpha k}(1-\alpha)^{(1-\alpha)k}} \right\rceil \geq 3$ and we always have a call in line 15). We can therefore think of the search tree of Algorithm 3 as an out-tree where all internal nodes have out-degree at least two and therefore the number of leaves is greater than the number of internal nodes.

Observe that each iteration of the for-loop in line 12 of Algorithm **find-tree** makes at most two recursive calls to **find-tree** and the time spent in each iteration of the for-loop is at most $O(n^2)$. As the time spent in each call of **find-tree** outside the for-loop is also bounded by $O(n^2)$ we obtain the desired complexity bound $O(n^2 k^{\rho} C^k)$.

Thus, it remains to show that $L(T, D) \leq R(k) = Bk^{\rho}C^k + 1$. First note that if $k = 0$ or $k = 1$ then line 1 is false exactly once (as there are no recursive calls) and $\min\{R(1), R(0)\} \geq 1 = L(T, D)$. If $k \in \{3, 4\}$, then line 1 is false a constant number of times by Lemma 4.2.7 and let B be the minimal integer such that $L(T, D) \leq R(k) = Bk^{\rho}C^k + 1$ for both $k = 3$ and 4. Thus, we may now assume that $k \geq 5$ and proceed by induction on k.

Let $R'(\alpha, k) = (6((1-\alpha)k)^{\rho}C^{(1-\alpha)k})/(\alpha^{\alpha k}(1-\alpha)^{(1-\alpha)k})$. Let α be defined as in line 8 of Algorithm **find-tree**. We will consider the following two cases separately.

Case 1, $\alpha \geq \alpha^*$: In this case we note that the following holds as $k \geq 2$ and $(1-\alpha) \geq \alpha$.

$$\begin{aligned} L(T, D) &\leq \left\lceil 2.51/(\alpha^{\alpha k}(1-\alpha)^{(1-\alpha)k}) \right\rceil \times (R(\alpha k) + R((1-\alpha)k)) \\ &\leq 3/(\alpha^{\alpha k}(1-\alpha)^{(1-\alpha)k}) \times (2 \cdot R((1-\alpha)k)) \\ &= R'(\alpha, k). \end{aligned}$$

By the definition of ρ we observe that $(1-\alpha^*)^{\rho} = 1/6$, which implies that the following holds by the definition of C:

$$R'(\alpha^*, k) = 6((1-\alpha^*)k)^{\rho}C^{(1-\alpha^*)k} \times C^{\alpha^* k} = k^{\rho}C^k = R(k).$$

Observe that

$$\ln(R'(\alpha,k)) = \ln(6) + \rho\,[\ln(k) + \ln(1-\alpha)] + k\,[(1-\alpha)\ln(C) - \alpha\ln(\alpha) - (1-\alpha)\ln(1-\alpha)]$$

We now differentiate $\ln(R'(\alpha,k))$ which gives us the following:

$$\begin{aligned} \tfrac{\partial(\ln(R'(\alpha,k)))}{\partial(\alpha)} &= \rho\tfrac{-1}{1-\alpha} + k\,(-\ln(C) - (1+\ln(\alpha)) + (1+\ln(1-\alpha))) \\ &= \tfrac{-\rho}{1-\alpha} + k\left(\ln\left(\tfrac{1-\alpha}{\alpha C}\right)\right). \end{aligned}$$

Since $k \geq 0$ we note that the above equality implies that $R'(\alpha,k)$ is a decreasing function in α in the interval $\alpha^* \leq \alpha \leq 1/2$. Therefore $L(T,D) \leq R'(\alpha,k) \leq R'(\alpha^*,k) = R(k)$, which proves Case 1.

Case 2, $\alpha < \alpha^*$: In this case we will specify the splitting vertex when we make recursive calls using the larger of U_w and U_b (defined in line 5 of Algorithm **find-tree**). Let α' denote the α-value in such a recursive call. By Lemma 4.2.5 we note that the following holds:

$$\frac{1}{2} \geq \alpha' \geq \frac{1-2\alpha}{1-\alpha} > \alpha^*.$$

Analogously to Case 1 (as $R'(\alpha',(1-\alpha)k)$ is a decreasing function in α' when $1/2 \geq \alpha' \geq \alpha^*$) we note that the L-values for these recursive calls are bounded by the following, where $\beta = \frac{1-2\alpha}{1-\alpha}$ (which implies that $(1-\alpha)(1-\beta) = \alpha$):

$$\begin{aligned} R'(\alpha',(1-\alpha)k) &\leq R'\,(\beta,(1-\alpha)k) \\ &= 3/\left(\left(\beta^{\beta}(1-\beta)^{(1-\beta)}\right)^{(1-\alpha)k}\right)\times 2\times R((1-\beta)(1-\alpha)k) \\ &= 6R(\alpha k)/\left(\left(\beta^{\beta}(1-\beta)^{(1-\beta)}\right)^{(1-\alpha)k}\right). \end{aligned}$$

Thus, in the worst case we may assume that $\alpha' = \beta = (1-2\alpha)/(1-\alpha)$ in all the recursive calls using the larger of U_w and U_b. The following now holds (as $k \geq 2$).

$$\begin{aligned} L(T,D) &\leq \left\lceil 2.51/(\alpha^{\alpha k}(1-\alpha)^{(1-\alpha)k})\right\rceil \times (R(\alpha k) + R'(\alpha',(1-\alpha)k)) \\ &\leq 3/(\alpha^{\alpha k}(1-\alpha)^{(1-\alpha)k}) \times R(\alpha k) \times \left(1 + 6/\left(\left(\beta^{\beta}(1-\beta)^{(1-\beta)}\right)^{(1-\alpha)k}\right)\right) \\ &\leq 3R(\alpha k)/(\alpha^{\alpha k}(1-\alpha)^{(1-\alpha)k}) \times 7/\left(\left(\beta^{\beta}(1-\beta)^{(1-\beta)}\right)^{(1-\alpha)k}\right) \end{aligned}$$

Let $R^*(\alpha,k)$ denote the bottom right-hand side of the above equality (for any value of α). By the definition of ρ we note that $\rho = \frac{2\ln(1/6)}{2\ln(1-\alpha^*)} = \frac{\ln(1/36)}{\ln(\alpha^*)}$, which implies that $(\alpha^*)^{\rho} = 1/36$. By the definition of C and the fact that if $\alpha = \alpha^*$ then $\beta = (1-2\alpha^*)/(1-\alpha^*) = \alpha^*$, we obtain the following:

$$\begin{aligned} R^*(\alpha^*,k) &= 3R(\alpha^*k)/(\alpha^{*\alpha^*k}(1-\alpha^*)^{(1-\alpha^*)k}) \times 7/\left(\left(\alpha^{*\alpha^*}(1-\alpha^*)^{(1-\alpha^*)}\right)^{(1-\alpha^*)k}\right) \\ &= 21 \cdot R(\alpha^*k) \cdot C^{\alpha^*k} \cdot C^{\alpha^*(1-\alpha^*)k} \\ &= 21\alpha^{*\rho}k^{\rho}C^{\alpha^*k} \times C^{(2\alpha^*-\alpha^{*2})k} \\ &= 21\alpha^{*\rho}R(k) \\ &< R(k). \end{aligned}$$

We will now simplify $R^*(\alpha,k)$ further, before we differentiate $\ln(R^*(\alpha,k))$. Note that $\beta = \frac{1-2\alpha}{1-\alpha}$ implies that $(1-\alpha)(1-\beta) = \alpha$ and $\beta(1-\alpha) = 1-2\alpha$.

$$\begin{aligned} R^*(\alpha,k) &= 21R(\alpha k)/(\alpha^{\alpha k}(1-\alpha)^{(1-\alpha)k}) \times 1/\left(\left(\beta^{\beta}(1-\beta)^{(1-\beta)}\right)^{(1-\alpha)k}\right) \\ &= 21(\alpha k)^{\rho}C^{\alpha k}/(\alpha^{\alpha k}(1-\alpha)^{(1-\alpha)k}) \times 1/\left(\left(\tfrac{1-2\alpha}{1-\alpha}\right)^{(1-2\alpha)k}\left(\tfrac{\alpha}{1-\alpha}\right)^{\alpha k}\right) \\ &= 21(\alpha k)^{\rho}\left(C^{\alpha}/(\alpha^{2\alpha}(1-2\alpha)^{(1-2\alpha)})\right)^{k}. \end{aligned}$$

Thus, we have the following:

$$\ln(R^*(\alpha,k)) = \ln(21) + \rho\,(\ln(k) + \ln(\alpha)) + k\,(\alpha\ln(C) - 2\alpha\ln(\alpha) - (1-2\alpha)\ln(1-2\alpha))\,.$$

We now differentiate $\ln(R^*(\alpha,k))$ which gives us the following:

$$\begin{aligned} \tfrac{\partial(\ln(R^*(\alpha,k)))}{\partial(\alpha)} &= \tfrac{\rho}{\alpha} + k\,(\ln(C) - 2(1+\ln(\alpha)) + 2(1+\ln(1-2\alpha))) \\ &= \tfrac{\rho}{\alpha} + k\left(\ln\left(\tfrac{C(1-2\alpha)^2}{\alpha^2}\right)\right) \end{aligned}$$

Since $k \geq 0$ we note that the above equality implies that $R^*(\alpha,k)$ is an increasing function in α in the interval $1/3 \leq \alpha \leq \alpha^*$. Therefore $L(T,D) \leq R^*(\alpha,k) \leq R^*(\alpha^*,k) < R(k)$, which proves Case 2. □

Theorem 4.2.10. *There is an $O(n^2 5.704^k)$ time randomized algorithm that solves the k-*Out-Tree *problem.*

4.3 Derandomization

In this section we discuss the derandomization of the algorithm **find-tree** using the general method presented by Chen et al. [31] and based on the construction of (n,k)-universal sets studied in [93].

Definition 4.3.1. *An (n,k)-universal set $\mathcal{F}$ is a set of functions from $[n]$ to $\{0,1\}$, such that for every subset $S \subseteq [n], |S| = k$ the set $\mathcal{F}|_S = \{f|_S : f \in \mathcal{F}\}$ is equal to the set 2^S of all the functions from S to $\{0,1\}$.*

Such an universal set can play in **find-tree** the role of the random colorings. In the same article [31], Chen et al. also give an algorithm to generate one :

Proposition 4.3.2. *([31]) There is an $O(n2^{k+12\log^2 k})$ time deterministic algorithm that constructs an (n, k)-universal set of size bounded by $n2^{k+12\log^2 k+2}$.*

Using this universal set alone, however, would not enable us to obtain a deterministic fixed-parameter algorithm for **find-tree**, as the size of the family (and, thus, the number of iterations in the main loop of the algorithm) would now also depend on n, besides k. Hence, Chen et al. make use (see [31]) of a family of pre-coloring functions $(g_{n,k,z})_{z\leq 2n}$ to obtain a fixed-parameter algorithm. To explain it, let us first give a result from Fredman et al. [59].

Proposition 4.3.3. *Let n and k be integers, $n \geq k$, and let q_0 be the smallest prime number such that $n \leq q_0 < 2n$. For any k-subset S in $Z_n = \{0, \ldots, n-1\}$, there is an integer z, $0 \leq z \leq q_0$, such that the function $g_{n,k,z}$ over Z_n, defined as $g_{n,k,z}(a) = (az \mod q_0) \mod k^2$, is injective from S.*

By the above proposition, computing a (k^2, k)-universal set $\mathcal{F}_{k^2,k}$ instead of a (n, k)-universal set is enough for our purposes. Indeed, if we are looking for a k-subgraph S in our graph, there exists $1 \leq z < 2n$ such that $g_{n,k,z}$ is injective on S, thus ensuring that the family $\mathcal{F}'_{k,n,z} = \mathcal{F}_{k^2,k} \circ g_{n,k,z} = \{f \circ g_{n,k,z} : f \in \mathcal{F}_{k^2,k}\}$ is such that $\mathcal{F}'_{k,n,z}|_S$ is equal to the set 2^S.

This way, derandomizing **find-tree** amounts to running it at most $2n$ times (once for each possible value of z), each time using as a set of coloring functions the family $\mathcal{F}'_{k,n,z}$. Two lines of the algorithm will then need to be modified :

```
12 for each function $f \in F'_{k,n,z}$ do
13 $\forall i$ such that $x_i \in V(D) - \bigcup_{u\in L} X_u$, let $v_i$ be colored in white if $f(i) = 0$ and in black
   if $f(i) = 1$
```

Besides, we also need to pre-compute a (k^2, k')-universal set for any $k' \leq k$, as this will be needed in the recursions steps of the algorithm. By Proposition 4.3.2, this can be done in time $O(k^3 2^{k+12\log^2 k})$. Note that these modifications make the algorithm **find-tree** deterministic.

Then, from Proposition 4.3.3 we deduce that if a digraph D contains an out-tree T meeting the requirements, then there exists a z such that $g_{n,k,z}$ is injective on $V(T)$. During the iteration of the algorithm corresponding to z there will be an $f \in \mathcal{F}'_{k,n,z}$ such that the vertices corresponding to U_w in D with be colored in white while the vertices corresponding to U_b will be colored in black. Using induction on k, we can prove that this deterministic algorithm correctly returns the required out-tree provided that such an out-tree exists in the digraph.

Let us briefly sketch how the running time is derived. We consider the following type of recurrence relations:

$$T(k, n) \leq X_0 2^k \times (T((1 - \alpha)k, n) + T(\alpha k, n))$$

Here X_0 is a constant determined by the size of the initial out-tree we are considering, and it adds to the exponent of $T(k, n)$ with $o(k)$ factor. On the other hand, the value of α asymptotically evolves around α^* as we see in the randomized version of algorithm. As a result, $T(k, n)$ is a function of the form $(2^{1/\alpha^*})^{k+o(k)}$. Overall the computation is similar to that described in the proof of Theorem 4.3.4. For completeness, we give a full proof.

Theorem 4.3.4. *Algorithm* **find-tree** *has running time $O(n^2 k^\rho C^k)$, where $w(T) = k$ and $|V(D)| = n$, and C and ρ are defined and bounded as follows:*

$$C = 2^{1/\alpha^*}, \; \rho = \frac{\log 1/(2X_0)}{\log (1 - \alpha^*)} \approx O(\log^2 k_0), \;\; \textit{and } C \leq 6.139.$$

Here, we set $X_0 = 2^{2 \log k_0 + 12 \log^2 k_0 + 2}$, where k_0 denotes the number of vertices in the initial tree.

Proof. Let $L(T, D)$ denote the number of times the 'if'-statement in line 1 of Algorithm **find-tree** is false (in all recursive calls to **find-tree**). We will prove that $L(T, D) \leq R(k) = Bk^\rho C^k + 1$, $B \geq 1$ is a constant whose value will determined later in the proof. This would imply that the number of calls to **find-tree** where the 'if'-statement in line 1 is true is also bounded by $R(k)$ as if line 1 is true then we will have at least two calls to **find-tree** (in fact it will have at least three as $k_0^2 2^{k+12\log^2 k+2} \geq 3$ and we always have a call in line 15). We can therefore think of the search tree of Algorithm 3 as an out-tree where all internal nodes have out-degree at least two and therefore the number of leaves is greater than the number of internal nodes.

Observe that each iteration of the for-loop in line 12 of Algorithm **find-tree** makes at most two recursive calls to **find-tree** and the time spent in each iteration of the for-loop is at most $O(n^2)$. As the time spent in each call of **find-tree** outside the for-loop is also bounded by $O(n^2)$ we obtain the desired complexity bound $O(n^2 k^\rho C^k)$.

Thus, it remains to show that $L(T, D) \leq R(k) = Bk^\rho C^k + 1$. First note that if $k = 0$ or $k = 1$ then line 1 is false exactly once (as there are no recursive calls) and $\min\{R(1), R(0)\} \geq 1 = L(T, D)$. If $k \in \{3, 4\}$, then line 1 is false a constant number of times by Lemma 4.2.7 and let B be the minimal integer such that $L(T, D) \leq R(k) = Bk^\rho C^k + 1$ for both $k = 3$ and 4. Thus, we may now assume that $k \geq 5$ and proceed by induction on k.

Let $R'(\alpha, k) = 2X_0B2^k((1-\alpha)k)^\rho(2^{1/\alpha^*})^{(1-\alpha)k}$. Let α be defined as in line 8 of Algorithm **find-tree**. We will consider the following two cases separately.

Case 1, $\alpha \geq \alpha^*$: In this case we note that the following holds as $k \geq 2$ and $(1-\alpha) \geq \alpha$. Note that we iterate the for-loops $k_0^2 2^{k+12\log^2 k+2} \leq X_0 2^k$ times in the derandomized algorithm.

$$\begin{aligned} L(T,D) &\leq X_0 2^k \times (R(\alpha k) + R((1-\alpha)k)) \\ &\leq X_0 2^k \times (2 \cdot R((1-\alpha)k)) \\ &= R'(\alpha, k). \end{aligned}$$

By the definition of ρ we observe that $(1-\alpha^*)^\rho = 1/(2X_0)$, which implies that the following holds by the definition of C:

$$R'(\alpha^*, k) = 2X_0B2^k((1-\alpha^*)k)^\rho(2^{1/\alpha^*})^{(1-\alpha^*)k} = Bk^\rho(2^{1/\alpha^*})^k = R(k).$$

Observe that

$$\ln(R'(\alpha,k)) = \ln(2X_0B) + \rho\,[\ln(k) + \ln(1-\alpha)] + k\,[\ln(2) + ((1-\alpha)/\alpha^*)\ln(2)]$$

We now differentiate $\ln(R'(\alpha,k))$ which gives us the following:

$$\frac{\partial(\ln(R'(\alpha,k)))}{\partial(\alpha)} = \rho\frac{-1}{1-\alpha} - \frac{k}{\alpha^*}\ln(2).$$

Since $k \geq 0$ we note that the above equality implies that $R'(\alpha,k)$ is a decreasing function in α in the interval $\alpha^* \leq \alpha \leq 1/2$. Therefore $L(T,D) \leq R'(\alpha,k) \leq R'(\alpha^*,k) = R(k)$, which proves Case 1.

Case 2, $\alpha < \alpha^*$: In this case we will specify the splitting vertex when we make recursive calls using the larger of U_w and U_b (defined in line 5 of Algorithm **find-tree**). Let α' denote the α-value in such a recursive call. By Lemma 4.2.5 we note that the following holds:

$$\frac{1}{2} \geq \alpha' \geq \frac{1-2\alpha}{1-\alpha} > \alpha^*.$$

Analogously to Case 1 (as $R'(\alpha', (1-\alpha)k)$ is a decreasing function in α' when $1/2 \geq \alpha' \geq \alpha^*$) we note that the L-values for these recursive calls are bounded by the following, where $\beta = \frac{1-2\alpha}{1-\alpha}$ (which implies that $(1-\alpha)(1-\beta) = \alpha$):

$$\begin{aligned} R'(\alpha', (1-\alpha)k) &\leq R'(\beta, (1-\alpha)k) \\ &= 2X_0 B 2^{(1-\alpha)k}((1-\beta)(1-\alpha)k)^{\rho}(2^{1/\alpha^*})^{(1-\beta)(1-\alpha)k} \\ &= 2X_0 B 2^{(1-\alpha)k}(\alpha k)^{\rho}(2^{1/\alpha^*})^{\alpha k} \\ &= 2X_0 2^{(1-\alpha)k} R(\alpha k) \end{aligned}$$

Thus, in the worst case we may assume that $\alpha' = \beta = (1-2\alpha)/(1-\alpha)$ in all the recursive calls using the larger of U_w and U_b. The following now holds (as $k \geq 2$).

$$\begin{aligned} L(T,D) &\leq X_0 2^k \times (R(\alpha k) + R'(\alpha', (1-\alpha)k)) \\ &\leq X_0 2^k \times \left(R(\alpha k) + 2X_0 2^{(1-\alpha)k} R(\alpha k))\right) \\ &\leq X_0 2^k \times R(\alpha k) \times \left(1 + 2X_0 2^{(1-\alpha)k}\right) \\ &\leq X_0 2^k \times R(\alpha k) \times 3X_0 2^{(1-\alpha)k} \end{aligned}$$

Let $R^*(\alpha, k)$ denote the bottom right-hand side of the above equality (for any value of α). By the definition of ρ we note that $\rho = \frac{2\ln(1/(2X_0))}{2\ln(1-\alpha^*)} = \frac{\ln(1/(4X_0^2)}{\ln(\alpha^*)}$, which implies that $(\alpha^*)^{\rho} = 1/(4X_0^2)$. By the definition of C and the fact that if $\alpha = \alpha^*$ then $\beta = (1-2\alpha^*)/(1-\alpha^*) = \alpha^*$, we obtain the following:

$$\begin{aligned} R^*(\alpha^*, k) &= X_0 2^k \times R(\alpha^* k) \times 3X_0 2^{(1-\alpha^*)k} \\ &= 3X_0^2 \times B(\alpha^* k)^{\rho}(2^{1/\alpha^*})^{\alpha^*} \times 2^{(2-\alpha^*)k} \\ &= \tfrac{3}{4} B k^{\rho} 2^{(3-\alpha^*)k} \\ &= \tfrac{3}{4} B k^{\rho} (2^{1/\alpha^*})^k \\ &< R(k) \end{aligned}$$

We will now simplify $R^*(\alpha, k)$ further, before we differentiate $\ln(R^*(\alpha, k))$.

$$R^*(\alpha, k) = 3X_0^2 B(\alpha k)^{\rho} 2^{k(2+\alpha/\alpha^* - \alpha)}$$

Thus, we have the following:

$$\ln(R^*(\alpha, k)) = \ln(3X_0^2 B) + \rho(\ln(k) + \ln(\alpha)) + k(2 + \frac{\alpha}{\alpha^*} - \alpha)\ln(2).$$

We now differentiate $\ln(R^*(\alpha, k))$ which gives us the following:

$$\frac{\partial(\ln(R^*(\alpha,k)))}{\partial(\alpha)} = \frac{\rho}{\alpha} + k(\frac{1}{\alpha^*} - 1)$$

Since $k \geq 0$ we note that the above equality implies that $R^*(\alpha, k)$ is an increasing function in α in the interval $1/3 \leq \alpha \leq \alpha^*$. Therefore $L(T,D) \leq R^*(\alpha, k) \leq R^*(\alpha^*, k) < R(k)$, which proves Case 2. □

Consequently we obtain the following.

Corollary 4.3.5. *There is an* $O(n^2 6.139^{k+o(k)}) = O(n^2 6.14^k)$ *time deterministic algorithm that solves the* k-OUT-TREE *problem.*

Chapter 5

Directed Minimum Leaf Problem

In this chapter we study the Directed Minimum Leaf problem. Given a digraph D, the Directed Minimum Leaf problem (MinLeaf) is the problem of finding an out-branching with the minimum possible number of leaves in D. Denote this minimum by $\ell_{\min}(D)$. When D has no out-branching, we write $\ell_{\min}(D) = 0$. Notice that not every digraph D has an out-branching. Nonetheless, we can easily decide whether D has an out-branching as Lemma 2.1.1 in Chapter 2 indicates and thus we may often assume that $\ell_{\min}(D) > 0$.

Since MinLeaf generalizes the hamiltonian directed path problem, MinLeaf is NP-hard. We will consider three parameterizations of MinLeaf: (a) $\ell_{\min}(D) \leq k$ $(k \geq 1)$, (c) $\ell_{\min}(D) \leq n/k$ $(k \geq 2)$, (d) $\ell_{\min}(D) \leq n - k$ $(k \geq 1)$, where n is the number of vertices in D and k is the parameter. We show that (a) and (b) are NP-complete for every value of the parameter, and the parameterization (c) will be our major concern.

We first consider MinLeaf restricted to acyclic digraphs in Section 5.1 and give a simple proof that MinLeaf can be solved in polynomial time in this case. In Section 5.2 we consider *nearly acyclic* digraphs and explore whether and how far the polynomial solvability of MinLeaf on acyclic instances can be extended. In the following sections we focus on the problem MinLeaf under the parameterization (c), which we call Directed k-Internal. In Section 5.4, we describe a kernelization producing quadratic order kernel for Directed k-Internal. Based on win/win strategy and the notion of tree decomposition, an fpt-algorithm of Directed k-Internal with running time $O(2^{O(k \log k)} + n^6)$ is presented in Section 5.5. Lastly we present an $O(c^k)$-algorithm for Directed k-Internal which use as a subroutine the fpt-algorithm for k-Out-tree from Chapter 4.

Throughout the chapter, the symbols n and m will denote the number of vertices and arcs in the digraph under consideration.

5.1 MinLeaf on Acyclic Digraphs

Let D be an acyclic digraph. We may assume that D has a unique vertex, r, of in-degree 0 as, by Lemma 2.1.1, this is a necessary and sufficient condition for D to have an out-branching. Let B be a bipartite graph with partite sets $X = V(D)$ and $X' = \{x' : x \in V(D) \setminus \{r'\}$ and edge set $E(B) = \{xy' : x \in X, y' \in X', xy \in A(D)\}$. Let $m(B)$ denotes the maximum size of a matching in B.

Lemma 5.1.1. *We have $\ell_{\min}(D) = |X| - m(B)$.*

Proof. A set N of edges of B is called *nice* if each vertex of X' is incident to exactly one edge in N and N contains an edge incident to r. Let T be an out-branching of D and let $f(T) = \{xy' : xy \in A(T)\}$. We will prove that $f : T \mapsto f(T)$ is a bijection between all out-branchings of D and all nice edge sets of B. Indeed, if P is an out-branching, then clearly $f(P)$ is a nice edge set. Let N be a nice edge set and let Q be a spanning subdigraph of D constructed as follows: $xy \in A(Q)$ if and only if $xy' \in N$. Since every vertex of X' is incident to exactly one edge of N, we have $d^-_Q(z) = 1$ for each $z \in V(Q) \setminus \{r\}$. Since Q is acyclic with a unique vertex of in-degree 0, Q is connected and, thus, Q is an out-branching. Clearly, $Q = f^{-1}(N)$.

Let T be an out-branching of D and let $B[f(T)]$ be the subgraph of B induced by the set $f(T)$. Observe that the number of leaves in T equals the number $iv(B[f(T)])$ of isolated vertices in $B[f(T)]$. Let N be a nice edge set in B, let $m(N)$ denote the maximum size of a matching in $B[N]$ and let H be a matching in $B[N]$ of size $m(N)$. Let $y' \in X'$ be a vertex of B not incident to an edge of H and let $xy' \in N$. Since H is maximum, x is incident to an edge of H. Thus, $iv(B[N]) = |X| - m(N)$ and $\ell_{\min}(D) = |X| - \max\{m(N) : N \text{ is nice}\}$.

Let M be a maximum matching in B and let M^* be obtained from M by adding to it an edge $uv' \in E(B)$ for each v' not covered by M. Notice that r is covered by M. Indeed, there exists a vertex u such that r is the only in-neighbor of u in D. Hence if r was not covered by M then u' would not be covered by M either, which means we could extend M by ru', a contradiction. Therefore, M^* covers r and, by definition, every vertex of X' is incident to exactly one edge of M^*. Thus, M^* is nice. Since $m(B) = m(M^*) = \max\{m(N) : N \text{ is nice}\}$, we conclude that $\ell_{\min}(D) = |X| - m(B)$. □

The correctness of the Algorithm 8 below follows from the proof of Lemma 5.1.1. The algorithm inputs an acyclic digraph D and outputs a minimum leaf out-branching T, if it exists, and 'NO', otherwise.

Let us analyze the computational complexity of Algorithm 8. Let n and m be the number of vertices and arcs in D. Each step of Algorithm 8 takes at most $O(m)$ time except for line 7. The computation time required to perform line 7 is the same as that of

Algorithm 7 Find a minimum leaf out branching on acyclic digraphs

```
if the number of vertices with in-degree 0 equals 1 then
    $r \leftarrow$ the vertex of in-degree 0
else
    return 'NO'
end if
Construct the bipartite graph $B$ of $D$
find a maximum matching $M$ in $B$ and set $M^* \leftarrow M$
for all $y' \in X'$ not covered by $M^*$ do
    $M^* \leftarrow M^* \cup \{$an arbitrary edge incident to $y'\}$
end for
$A(T) \leftarrow \emptyset$
for all $xy' \in M^*$ do $A(T) \leftarrow A(T) \cup \{xy\}$
return $T$
```

solving the maximum cardinality matching problem on a bipartite graph. The last problem can be solved in time $O(|V(B)|^{1.5}\sqrt{|E(B)|/\log|V(B)|})$ [9]. Hence, the algorithm requires at most $O(m + n^{1.5}\sqrt{m/\log n})$ time.

Thus, we have the following:

Theorem 5.1.2. *Let D be an acyclic digraph. Then the Algorithm 8 returns a minimum leaf out-branching if one exists, or returns 'NO' otherwise in time* $O(m + n^{1.5}\sqrt{m/\log n})$.

5.2 MinLeaf on Near-Acyclic Digraphs

In this section we investigate how far we can extend the polynomiality result for acyclic digraphs. Notice that acyclic digraphs are the digraphs of directed path-width (directed tree-width, DAG-width, respectively) 0. We prove that already for digraphs of directed path-width (directed tree-width, DAG-width, respectively) 1, MinLeaf is NP-hard. This is in sharp contrast to the fact that the Hamilton path problem (the most important special case of MinLeaf) is polynomial time solvable for digraphs of bounded directed path-width (directed tree-width, DAG-width, respectively).

On the other hand, the hardness of MinLeaf on near-acyclic digraphs does not exclude the possibility of MinLeaf, the problem of checking if there is an out-branching with at most k leaves, being polynomial-time solvable for *fixed k*. We shall subsequently show that for digraphs of bounded directed tree-width (directed path-width, DAG-width, respectively) and a fixed integer k, the problem of checking whether there is an out-branching with at most k leaves is polynomial time solvable.

In this section, we use the following linkage problem and its algorithm from [76]. Let

$$\sigma = (s_1, t_1, s_2, t_2, \ldots, s_p, t_p)$$

be a sequence of $2p$ vertices of a digraph D, (vertices in σ are not necessarily distinct). A *hamiltonian σ-linkage* of D is a collection of p directed paths $P_1, P_2, \ldots, P_p$ such that $V(P_1) \cup \ldots \cup V(P_p) = V(D)$, P_i starts at s_i and terminates at t_i, $1 \leq i \leq p$, and $(V(P_i) \setminus \{s_i, t_i\}) \cap (V(P_j) \setminus \{s_j, t_j\}) = \emptyset$ for all $1 \leq i < j \leq p$. In the *hamiltonian linkage problem*, given σ we are to check whether there is a hamiltonian σ-linkage of D.

The following lemma is well-known [14, 16, 76, 96] and easy to prove using just the definitions above.

Lemma 5.2.1. *Let D be a digraph. For* $\mathrm{d} \in \{\mathrm{dag}, \mathrm{dt}, \mathrm{dp}\}$*, we have* $\mathrm{dw}(D) = 0$ *if and only if D is acyclic.*

Lemma 5.2.2. *For a digraph D, we have* $\mathrm{dtw}(D) \leq \mathrm{dpw}(D)$.

Proof: Let $Y_1, Y_2, \ldots, Y_k$ be the bags in a DPD of D. We may assume that all bags are distinct. Define an arboreal decomposition of D, where the arborescence is the directed path $12\ldots k$, as follows: $W_1 = Y_1$, $W_i = Y_i \setminus Y_{i-1}$ for each $i = 2, 3, \ldots, k$ and if $e = (i, i+1)$ we let $X_e = Y_i \cap Y_{i+1}$. This arboreal decomposition is of the same width as the DPD and we are done. □

5.2.1 Hardness Result

Here we give a proof that MinLeaf is NP-hard for digraphs of directed path-width (directed tree-width, DAG-width, respectively) 1. If P is a directed path and vertices a, b are, in that order, on P, then we denote the $a-b$-segment of P by $P[a, b]$, and by $P[b, *]$ we mean the $b-t$-segment of P, where t is the terminal vertex of P.

Theorem 5.2.3. *MinLOB is NP-hard for digraphs of directed path-width (directed tree-width, DAG-width, respectively) 1.*

Proof: We prove the theorem by reduction of 3SAT to MinLOB. We use the following gadget H, the digraph with vertex set $V(H) = \{x_1, y_1, z_1, x_2, y_2, z_2\}$ and arc set $A(H) = \{x_1y_1, y_1z_1, z_1x_1, x_1x_2, y_1y_2, z_1z_2, x_2z_2, z_2y_2, y_2x_2\}$. It is easy to verify that H has the following properties:

(i) there exists a hamiltonian (x_1, x_2)-linkage P_x of H,

(ii) there exists a hamiltonian (x_1, x_2, y_1, y_2)-linkage of H,

(iii) there exists an hamiltonian $(x_1, x_2, y_1, y_2, z_1, z_2)$-linkage of H,

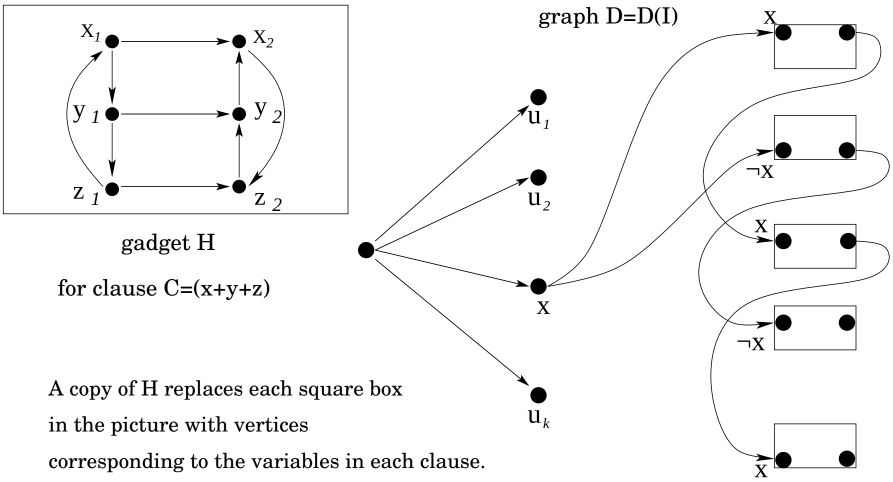
x1
x2
y 1
y 2
z 1
z 2
gadget H
for clause C=(x+y+z)
A copy of H replaces each square box
in the picture with vertices
corresponding to the variables in each clause.
graph D=D(I)
u1
u2
x
uk
x
¬x
x
¬x
x

Figure 5.1: The gadget H and construction of $D = D(I)$ from 3-SAT instance.

(iv) if P_x is a hamilton path of H starting at x_1 then P_x ends in x_2,
(v) if P_x and P_y are vertex disjoint paths in H starting at x_1 and y_1, respectively, which go through all vertices of H, then either P_x ends in x_2 and P_y ends in y_2, or P_x ends in y_2 and P_y ends in z_1.

Analogous statements hold for each permutation of x, y, z.

Consider an instance I of 3SAT with variables $v^1, v^2, \ldots, v^k$ and clauses $C_1, C_2, \ldots, C_p$. Construct a digraph $D = D(I)$ as follows: For each clause C_j let H_j be a copy of H. If $C = \alpha + \beta + \gamma$, where α,β, and γ are literals, denote the vertices of H_j by $\alpha_1(H_j)$,$\beta_1(H_j)$, $\gamma_1(H_j)$, $\alpha_2(H_j)$,$\beta_2(H_j)$, $\gamma_2(H_j)$. (Occasionally, when we do not wish to specify the variables α,β,γ, we denote the vertices simply by $x_1(H_j), \ldots, z_2(H_j)$.) We also introduce a vertex u_i for each variable v^i and a root vertex r. So

$$V(D) = \{r, u_1, u_2, \ldots, u_k\} \cup \bigcup_{j=1}^{p} V(H_j),$$

and D is a graph of order $6p + k + 1$.

The arc set of D consists of $\bigcup_{j=1}^{p} E(H_j)$, arcs ru_i for $i = 1, 2, \ldots, k$ and the arcs in the sets $\mathrm{Arc}(v^1), \mathrm{Arc}(\overline{v^1}), \ldots, \mathrm{Arc}(v^k), \mathrm{Arc}(\overline{v^k})$ defined as follows. Consider a variable v^i. Let $C_{j_1}, C_{j_2}, \ldots, C_{j_s}$, with $j_1 < j_2 < \ldots < j_s$, be the clauses containing v^i as literal. Then the set $\mathrm{Arc}(v^i)$ contains the arcs $u_iv^i_1(H_{j_1})$, $v^i_2(H_{j_1})v^i_1(H_{j_2})$, $v^i_2(H_{j_2})v^i_1(H_{j_3})$, $\ldots$, $v^i_2(H_{j_{s-1}})v^i_1(H_{j_s})$. Similarly let $C_{h_1}, C_{h_2}, \ldots, C_{h_t}$, with $h_1 < h_2 < \ldots < h_t$, be the clauses containing $\overline{v^i}$ as literal. Then the set $\mathrm{Arc}(\overline{v^i})$ contains the arcs $u_i\overline{v^i}_1(H_{h_1})$, $\overline{v^i}_2(H_{h_1})\overline{v^i}_1(H_{h_2})$, $\overline{v^i}_2(H_{h_2})\overline{v^i}_1(H_{h_3})$,$\ldots$, $\overline{v^i}_2(H_{h_t-1})\overline{v^i}_1(H_{h_t})$. This completes the construction of D.

We prove that

$$\mathrm{dtw}(D) = \mathrm{dagw}(D) = \mathrm{dpw}(D) = 1 \tag{5.1}$$

Since D is not acyclic, by Lemma 5.2.1, every width parameter in (5.1) is positive and, by Lemma 5.2.2, it is enough to show that $\mathrm{dpw}(D) \leq 1$. It can be easily checked that the following bags form a DPD of D of width 1:

$$\{r\}, \{u_1\}, \{u_2\}, \ldots, \{u_k\},$$

$$\{z_1(H_1), y_1(H_1)\}, \{y_1(H_1), x_1(H_1)\}, \{x_2(H_1), y_2(H_1)\}, \{y_2(H_1), z_2(H_1)\},$$

$$\ldots, \{z_1(H_p), y_1(H_p)\}, \{y_1(H_p), x_1(H_p)\}, \{x_2(H_p), y_2(H_p)\}, \{y_2(H_p), z_2(H_p).$$

We now show that D has an out-branching with exactly k leaves if and only if I is satisfiable.

Given a valid truth assignment to $v^1, \ldots, v^k$ we construct an out-branching B of D with k leaves as follows. Root B at r. Let $ru_1, ru_2, \ldots, ru_k \in E(B)$. If variable v^i has truth value TRUE then add all arcs in $\mathrm{Arc}(v^i)$ to $A(B)$. Then these arcs, together with suitably (i.e., according to properties (i), (ii) and (iii) of H) chosen $v^i_1(H_j) - v^i_2(H_j)$ paths through those H_j which correspond to the C_j containing v^i as a literal, yield a path $P(v^i)$ starting at u_i. Similarly, if variable v^i has truth value FALSE then add all arcs in $\mathrm{Arc}(\overline{v^i})$ and suitably chosen $\overline{v^i}_1(H_j) - \overline{v^i}_2(H_j)$ paths to $A(B)$ and obtain a path $P(\overline{v^i})$ starting at u_i. Since these k paths, attached to the vertices $u_1, \ldots, u_k$, go through all vertices in $V(D)$, B is an out-branching of D with exactly k leaves.

Given an out-branching B with exactly k leaves of D, we derive an assignment of truth values to the variables $v^1, \ldots, v^k$ that satisfies each clause C_j and thus I. We note that B must be rooted at r since $\mathrm{d}^-_D(r) = 0$ and that $ru_i \in A(B)$ for $i = 1, 2, \ldots, k$ since $\mathrm{d}^-_D(u_i) = 1$. So $\mathrm{d}^+_T(r) = k$, hence the subtree of T rooted at u_i is a path P_i for $i = 1, 2, \ldots, k$.

Consider a subgraph H_j of D. A path P_i that intersects with H_j is said to be H_j-compatible if P_i enters H_j at x_1 and leaves at x_2, or it enters H_j at y_1 and leaves at y_2, or it enters H_j at z_1 and leaves at z_2. We now show that B can be modified, without changing the number of leaves, so that whenever a path P_i and a gadget H_j intersect, P_i is H_j-compatible. Consider a fixed H_j. First assume that P_i is the only path that intersects H_j. By property (iv) P_i is H_j-compatible. Next assume that two paths, P_h and P_i say, intersect H_j and that they enter H_j in, say, x_1 and y_1, respectively. By property (v) either P_h and P_i are H_j-compatible, or P_i ends in z_1 and P_h ends in y_2. In the latter case let P'_h be the union of $P_h[u_h, x_1]$ and the path x_1, x_2, and let P'_i be the union of $P_i[u_i, y_1]$, the path y_1, z_1, z_2, y_2 and $P_h[y_2, *]$, and replace P_h and P_i by P'_h and P'_i. Finally assume that three paths P_g, P_h, P_i intersect H_j. Then a similar construction yields H_j-compatible paths

P'_g, P'_h and P'_i. Clearly, replacing P_g, P_h, P_i by P'_g, P'_h, P'_i if necessary does not change the number of leaves of B, nor does it create any incompatibilities. Hence repeating this step for all H_j eventually yields an out-branching in which every path P_i that intersects a gadget H_j is H_j-compatible.

Note that vertex u_i has two out-neighbors in D, $v^i_1(H_{j_1})$ and $\overline{v^i}_1(H_{h_1})$, where C_{j_1} (C_{h_1}) is the first clause to contain v^i $(\overline{v^i})$ as a literal, and that T contains at most one of these arcs. If the first arc of P_i is $u_i v^i_1(H_{j_1})$ then we assign the value TRUE to v^i, if the first arc of P_i is $u_i\overline{v^i}_1(H_{h_1})$ then we assign the value FALSE to v^i, and if P_i has no arc we assign an arbitrary truth value to v^i. It remains to show that this satisfies I.

Fix an arbitrary clause C_j and consider H_j. There is at least one path P_i of the out-branching B that intersects with H_j. Assume that the first arc of P_i is, say, $u_i v^i_1(H_{j_1})$ (for $u_i\overline{v^i}_1(H_{h_1})$ the proof is analogous) and that P passes through $H_{j_1}, H_{j_2}, \ldots$ before reaching H_j. Since P_i is compatible with $H_{j_1}, H_{j_2}, \ldots, H_j$, it enters $H_{j_1}, H_{j_2}, \ldots, H_j$ in $v^i_1(H_{j_1})$, $v^i_1(H_{j_2}), \ldots, v^i_1(H_j)$. Hence clauses $C_{j_1}, C_{j_2}, \ldots, C_j$ contain v^i as a literal. But since we assigned the value TRUE to v^i, clause C_j is satisfied. Since C_j was arbitrary, all clauses and thus I are satisfied. □

5.2.2 Polynomial Algorithm For Fixed Number of Leaves

We need the following theorem for our polynomiality result in this subsection.

Theorem 5.2.4. *[76] For every fixed positive integer p and every fixed nonnegative integer w the hamiltonian linkage problem with input sequence σ of $2p$ vertices for digraphs of directed tree-width at most w is polynomial time solvable.*

Theorem 5.2.5. *Let* $\mathrm{d} \in \{\mathrm{dag}, \mathrm{dt}, \mathrm{dp}\}$. *For every fixed positive integer k and every fixed nonnegative integer w, we can check, in polynomial time, whether a digraph D with* $\mathrm{dw}(D) \le w$ *has an out-branching with at most k leaves.*

Proof: Let D be a digraph. By Lemma 5.2.2, if $\mathrm{dpw}(D) \le w$ then $\mathrm{dtw}(D) \le w$. It is shown in [16] that if $\mathrm{dagw}(D) \le w$ then $\mathrm{dtw}(D) \le 3w + 1$.

Thus, we may assume that D is of directed tree-width at most w, for some integer w, and let B be an out-branching in D with at most k leaves. Let $X(B)$ be the set consisting of the root, the leaves and the branching vertices of B. It is not difficult to show that $|X(B)| \le 2k$. Now contract each directed path of B between two vertices of $X(B)$ into an arc (between the vertices of $X(B)$) and observe that we have obtained an out-tree B' with exactly $|X(B)|$ vertices. We call B' the *contraction* of B.

Now let $Y \subseteq V(D)$, $|Y| \le 2k$, and let T be an out-branching constructed on the vertices of $D[Y]$ with arcs $A(T) = \{(s_1, t_1), (s_2, t_2), \ldots, (s_{|Y|-1}, t_{|Y|-1})\}$. Notice here that the arcs of

$A(T)$ may be not present in the digraph D. Using the algorithm of Theorem 5.2.4 with input $(s_1, t_1, s_2, t_2, \dots, s_{|Y|-1}, t_{|Y|-1})$, we can check, in polynomial time, whether D contains an out-branching B^* whose contraction is T.

Thus, to find an out-branching in D with the minimum number of leaves, we can use the following procedure. We generate all subsets of $V(D)$ with at most $2k$ vertices and, for each such subset Y, we generate all out-branchings T in $D[Y]$. For each T we use the algorithm of Theorem 5.2.4 to verify whether D has an out-branching whose contraction is T. Finally, we find a minimum leaf out-branching among all the outputs of the algorithm if one with at most k leaves exists.

Observe that for each Y, by Cayley's formula on the number of spanning trees in a complete graph, there are at most $|Y|^{|Y|-1}$ out-branchings of $D[Y]$ and that there are less than $|V(D)|^{2k+1}$ sets Y with $|Y| \leq 2k$. Thus, in our procedure, we use the algorithm of Theorem 5.2.4 less than $|V(D)|^{2k+1} \cdot (2k)^{2k-1}$ times, which shows that the running time of the procedure is polynomial. □

5.3 Three Parameterizations of MinLeaf

The following is a natural way to parameterize the problem MinLeaf.

MinLeaf Parameterized Naturally
Instance: A digraph D.
Parameter: A positive integer k.
Question: Is $\ell_{\min}(D) \leq k$?

Clearly, this problem is NP-complete already for $k = 1$ as for $k = 1$ it is equivalent to the hamiltonian directed path problem. Let v be an arbitrary vertex of D. Transform D into a new digraph D_k by adding k vertices $v_1, v_2, \dots, v_k$ together with the arcs $vv_1, vv_2, \dots, vv_k$. Observe that D has a hamiltonian directed path terminating at v if and only if $\ell_{\min}(D_k) \leq k$. Since the problem is NP-complete of checking whether a digraph has a hamiltonian directed path terminating at a prescribed vertex, we conclude that under the standard parameterization the problem MinLeaf is NP-complete for every fixed k.

Let us denote by $\vec{K}_{1,p-1}$ the *star digraph* of order p, i.e., the digraph with vertices $1, 2, \dots, p$ and arcs $12, 13, \dots, 1p$. We consider a slightly weaker parameterization of MinLeaf.

MinLeaf Parameterized Strongly Below Guaranteed Value
Instance: A digraph D of order n with $\ell_{\min}(D) > 0$.
Parameter: An integer $k \geq 2$.
Question: Is $\ell_{\min}(D) \leq n/k$?

Under this parameterization the problem MinLeaf is still NP-complete for every fixed $k \geq 2$. To prove this consider a digraph D of order n and a digraph H obtained from D by adding to it the star digraph $\vec{K}_{1,p-1}$ on $p = \lfloor n/(k-1) \rfloor$ vertices ($V(D) \cap V(\vec{K}_{1,p-1}) = \emptyset$) and appending an arc from vertex 1 of $\vec{K}_{1,p-1}$ to an arbitrary vertex y of D. Observe that $\ell_{\min}(H) = p - 1 + \ell_{min}(D, y)$, where $\ell_{min}(D, y)$ is the minimum possible number of leaves in an out-branching rooted at y, and that $\frac{1}{k}|V(H)| = p + \varepsilon$, where $0 \leq \varepsilon < 1$. Thus, $\ell_{\min}(H) \leq \frac{1}{k}|V(H)|$ if and only if $\ell_{min}(D, y) = 1$. Hence, the hamiltonian directed path problem with fixed initial vertex (vertex y in D) can be reduced to this MinLeaf *parameterized strongly below guaranteed value* for every fixed $k \geq 2$ and, therefore, it is NP-complete for every $k \geq 2$.

Consider the following parameterizations of MinLeaf.

> **MinLeaf Parameterized Below Guaranteed Value** Directed k-Internal
> *Instance:* A digraph D of order n with $\ell_{\min}(D) > 0$.
> *Parameter:* A positive integer k.
> *Question:* Is $\ell_{\min}(D) \leq n - k$?
> *Solution:* An out-branching B of D with at most $n - k$ leaves or the answer 'NO' to the above question.

In the rest of the chapter we will consider this last parameterization of MinLeaf. Observe that we can equivalently state MinLeaf under the last parameterization as the problem of finding an out-branching with *at least k internal* vertices, if one exists. Henceforth, MinLeaf parameterized below guaranteed value will be denoted as Directed k-Internal.

5.4 Quadratic Kernel for Directed k-Internal

In this section we introduce a reduction rule for the Directed k-Internal problem. Using the reduction rule we present a polynomial time algorithm that either yields an out-branching with at most $n - k$ leaves or produces a kernel whose order is bounded by a quadratic function of k.

Let T be an out-branching of a given digraph D and let $(u, v) \in A(D) \setminus A(T)$. We define the *1-change* for (u, v) as the operation to add the arc (u, v) to T and remove the existing arc $(p(v), v)$ from T, where $p(v)$ is the *parent* (i.e. in-neighbor) of v in T. We say an out-branching is *minimal* if no 1-change for an arc of $A(D) \setminus A(T)$ leads to an out-branching with more internal vertices, or equivalently, less leaves. For two vertices x, y, we write $x \leq_T y$ if there is a path from x to y in T and especially when $x \neq y$, we write $x <_T y$. An arc $(y, x) \in A(D) \setminus A(T)$ is T-backward if $x <_T y$. The following is a simple observation on a minimal out-branching.

Lemma 5.4.1. *Let T be an out-branching of D. Then T is minimal if and only if for every arc $(u, v) \in A(D) \setminus A(T)$ which is not T-backward arc, the vertex u is internal or $d^+(p(v)) = 1$.*

Proof. Suppose the 1-change for $(u, v) \in A(D) \setminus A(T)$ yields an out-branching with less leaves. It is easy to see that (u, v) is not T-backward, u is a leaf and $d^+(p(v)) \geq 2$. Conversely if there is an arc $(u, v) \in A(D) \setminus A(T)$ which is not T-backward, u is a leaf and $d^+(p(v)) \geq 2$ then 1-change for (u, v) produces an out-branching in which the number number of leaves is strictly decreased. □

Lemma 5.4.2. *Given a digraph D, we can either build a minimal out-branching T with at most $n - k$ leaves or obtain a vertex cover of size at most $2k - 2$ in $O(n^2m)$ time.*

Proof. Let T be a minimal out-branching. If T has at most $n - k$ leaves, we are done. Suppose it is not. We claim that the set $U = \{u \in V(D) : u \text{ is internal in } T\} \cup \{u \in V(D) : u \text{ is a leaf in } T \text{ and } d^+(p(u)) = 1\}$ is a vertex cover of D. Since the set of internal vertices cover all arcs which are not between the leaves, it suffices to show that every arc (u, v) between two leaves u and v is covered by U. The last statement follows from the fact that T is minimal and Lemma 5.4.1. What remains is to observe that the number of internal vertices is at most $k - 1$ and the number of leaves which is the only child of its parent is at most $k - 1$ as well.

Now we consider the time complexity of the algorithm. The construction of an out-branching T of D takes $O(n + m)$ time. Whether T is minimal can be checked in $O(nm)$ time since for every arc $(u, v) \in A(D) \setminus A(T)$ we test the conditions of Lemma 5.4.1. Let L be the list of arcs $(u, v) \in A(D) \setminus A(T)$ which violates the minimality of T, i.e. such that u is a leaf and $d^+(p(v)) \geq 2$. Whenever $L \neq \emptyset$, choose $(u, v) \in L$ and transform T by replacing the arc $(p(v), v)$ by (u, v). Accordingly we update the list L as follows: (1) erase all arcs whose tail is u, which takes $O(m)$ time (2) erase all arcs whose head is v, which takes $O(m)$ time (3) add to L arcs of the form (x, y) where x is a leaf of the subtree rooted at v and y is a vertex with $d^+(p(y)) \geq 2$ on the unique path from the root of T to $p(v)$. This takes $O(nm)$ time. The validation of the update with (1)-(3) can be easily verified. Since any out -branching has at least one leaf and we decrease the number of leaves of T by 1 at each transformation, after at most n such transformations we obtain an out-branching where no further transformation can be done. This will be our minimal out-branching. When the minimal out-branching has more than $n - k$ leaves, we can construct the vertex cover U as above in $O(n)$ time. □

It follows from Lemma 5.4.2 that we can find either an out-branching which certifies a positive answer for the Directed k-Internal problem or a vertex cover of D of size at most $2k - 2$. In the second case, we can remove some redundant vertices from the large independent set of size at least $n - (2k - 2)$ and obtain an instance of smaller size. The *crown structure* plays the fundamental role in this reduction.

Definition 5.4.3. *A crown in a graph G is a pair (H, C), where $H \subseteq V(G)$ and $C \subseteq V(G)$ with $H \cap C = \emptyset$ such that the following conditions hold:*
(a) The set of neighbors of vertices in C is precisely H, i.e. $H = N(C)$,
(b) $C = C_m \cup C_u$ is an independent set, and
(c) There is a perfect matching between C_m and H.

A crown structure is a relatively new idea that allows us to have powerful reduction rules.

Given a digraph D, let U be a vertex cover of D. Modify U by adding to it the vertex of in-degree 0 if one exists. Let $W = V(D) \setminus U$ and observe that W is an independent set. We define the *internal number* of D as the largest possible number of internal vertices of an out-branching of D.

In order to accommodate a crown structure to Directed k-Internal we create an auxiliary model. Given a directed graph D with U and W as above, we build the (undirected) bipartite graph B as follows.

- $V(B) = U' \cup W$, where $U' = N^-(W) \cup (U \times U)$.
- $E(B) = \{\{xy, w\} : xy \in U \times U, w \in W, (x, w) \in A(D), (w, y) \in A(D)\} \cup \{\{x, w\} : x \in U, w \in W, (x, w) \in A(D)\}$

Observe that $N^-(W) \subseteq U$ as U is a vertex cover of D and that no vertex of W in B is isolated since every vertex of W is of in-degree at least one in D.

Lemma 5.4.4. *If B contains a crown $(H, C = C_m \cup C_u)$ with $C \subseteq W$ and $C_u \neq \emptyset$, then the internal number of D equals the internal number of $D - C_u$.*

Proof. We can extend an out-branching T of $D - C_u$ by appending an arc $(x, w) \in A(D)$, where $w \in C_u$ and x is any in-neighbor of w. The attachment of such an arc does not decrease the number of internal vertices of T. This shows that the internal number of D is not smaller than that of $D - C_u$.

Let a crown $(H, C = C_m \cup C_u)$ with $C \subseteq W$ and a perfect matching M between H and C_m are given. We start with the following claim.

Claim 1. Let c_{root} be the root of T. If $c_{root} \in C$, we can modify the perfect matching M into M' between H and $C'_m \subseteq C$ so that $c_{root} \in C'_m$ and $\{ux, c_{root}\} \in M'$ for some pair vertex $ux \in U \times U$.

Proof of Claim 1. Suppose this is not the case. Recall that c_{root} is of in-degree at least 1 since we excluded any vertex of in-degree 0 from W. Let u be an in-neighbor of c_{root} in D and x be a child of c_{root} in T. Note that $\{u, c_{root}\}, \{ux, c_{root}\} \in E(B)$ and thus $u, ux \in H$.

There are two cases and for each case we can obtain a new perfect matching as follows. Firstly if $c_{root} \in C_u$, simply exchange it with a vertex $c \in C_m$ which is matched to the pair vertex ux by M. This exchange is justified since $\{ux, c_{root}\} \in E(B)$. Secondly suppose $c_{root} \in C_m$ but it is matched to a vertex $u \in N^-(W)$. Since $(u, c_{root}), (c_{root}, x) \in A(D)$, we have the pair vertex ux in U' and moreover it is in H. Hence we can find $c \in C_m$ which is matched to the pair vertex ux and by exchanging it with c_{root} we have a new perfect matching. This is possible as we have $\{ux, c_{root}\} \in E(B)$ and $(u, c) \in A(D)$, thus $\{u, c\} \in E(B)$. □

Due to Claim 1, when $c_{root} \in C$ we may always assume that $c_{root} \in C_m$ and furthermore that $\{ux, c_{root}\} \in M$ for some pair vertex $ux \in (U \times U)$. Notice that x is not necessarily a child of c_{root} in T.

We shall show that the internal number of $D - C_u$ is not smaller than the internal number of D. To see this suppose T is an out-branching of D and consider the subgraph $F = T - C$ obtained from T by deleting the vertices of C. Obviously F is a union of out-trees, say $F_1, \dots, F_l$. We will add the vertices of C_m and a set of arcs so that we obtain an out-branching of $D - C_u$ with as many internal vertices as in T at the end of this process.

Recalling that $C \subseteq W$ is an independent set, it is straightforward to see any vertex $c \in C$ falls into one of the three types: (a) c is a leaf in T hanging to some vertex of F (b) c is an internal vertex in T which has both a parent and children in F (c) c is the root c_{root} of T and it has at least one in-neighbor in $V(D)$.

Let $c_1, \dots, c_t \in C$ be the vertices that are of type (b) in T. For each c_i, $1 \leq i \leq t$, let $H_i = \{f_p f_q \in U \times U : (f_p, c_i) \in A(T), (c_i, f_q) \in A(T)\}$. We denote $\bigcup_{1 \leq i \leq t} H_i$ by H_{int}. For the vertex $c_{root} \in C$, let $H_{root} = \{f_p x \in U \times U : (f_p, c_{root}) \in A(D) \setminus A(T), (c_{root}, x) \in A(T)\}$. We set $H_{root} = \emptyset$ if $c_{root} \notin C$. Note that both H_{int} and H_{root} belong to H.

The following procedure defines how to construct an out-tree T'' from F. We initialize $T' \leftarrow F$ and $C_{int} \leftarrow \emptyset$.

```
1. For every $f_p f_q \in H_{int}$
    1.1 let $H_i$ be the unique set containing $f_p f_q$.
    1.2 let $c_{pq} \in C_m$ be the vertex with $\{f_p f_q, c_{pq}\} \in M$
    1.3 $T' \leftarrow T' + c_{pq} + (f_p, c_{pq}) + (c_{pq}, f_q)$.
    1.4 $C_{int} \leftarrow C_{int} \cup \{c_{pq}\}$.
2. $T'' \leftarrow T'$.
3. If $c_{root} \notin C$, return $T''$.
```

```
4. If $c_{root} \notin C_{int}$
    4.1 $T'' \leftarrow T'' + c_{root}$.
    4.2 for each child $x$ of $c_{root}$ in $T$, $T'' \leftarrow T'' + (c_{root}, x)$.
    4.3 return $T''$.
5. Otherwise
    5.1 let $f_p f_q \in H_{int}$ be the vertex with $\{f_p f_q, c_{root}\} \in M$.
    5.2 let $x$ be the child of $c_{root}$ in $T$ with $x \leq_{T''} f_p$.
    5.3 let $c_x \in C_m$ be the vertex with $\{f_p x, c_x\} \in M$.
    5.4 $T'' \leftarrow T'' + c_x + (c_x, x)$.
    5.5 for each child $y \neq x$ of $c_{root}$ in $T$ (if any)
    5.5.1 let $c_y \in C_m$ be the vertex with $\{f_p y, c_y\} \in M$
    5.5.2 $T'' \leftarrow T'' + c_y + (f_p, c_y) + (c_y, y)$.
    5.6 return $T''$
```

Claim 2. Step 1 is valid and T' at step 2 is a union of out-trees.

Proof of Claim 2. For each $f_p f_q \in H_{int}$, the vertex $f_q \in V(F)$ appears as the second element of the pair vertex in H_{int} at most once. The uniqueness of $H_i \ni f_p f_q$ then follows (step 1.1). Moreover by the construction of H_i, $\{f_p f_q, c_i\} \in E(B)$ and thus $f_p f_q \in N(C) = H$, where the last equality follows by the definition of crown. Hence $f_p f_q$ is uniquely matched to a vertex $c_{pq} \in C_m$ by M (step 1.2). Also $\{f_p f_q, c_{pq}\} \in E(B)$ implies $(f_p, c_{pq}), (c_{pq}, f_q) \in A(D)$, which implies that T' can be properly constructed (step 1.3).

Now observe that any second element f_q of a pair vertex $f_p f_q \in H_{int}$ is a root of an out-tree in F. Thus for each component F_q of F, T' contains at most one arc entering into its root. Moreover, $f_p <_{T'} f_q$ if and only if $f_p <_T f_q$, which means there is no directed cycle in T'. Witnessing that all the other vertices have at most one arc entering into it, we conclude T' at step 2 is a union of out-trees. □

We claim that the above procedure returns an out-tree T''

Claim 3. Steps 3-5 are valid and T'' is an out-tree.

Proof of Claim 3. First consider the case when T'' is returned at step 3. With Claim 2, it is enough to show that T' is connected. Let two components F_p and F_q in F be connected by c_i in T. Since $c_{root} \notin C$, the vertex c_i is of type (b) and thus there exist $f_p \in F_p$ and the root f_q of F_q such that $(f_p, c_i) \in A(T)$, $(c_i, f_q) \in A(T)$. By the construction of H_{int}, we have $f_p f_q \in H_i \subseteq H_{int}$ and the vertex $c_{pq} \in C_m$ with $\{f_p f_q, c_{pq}\} \in M$ connects F_p and F_q in T' during the performance of step 1. Hence T' is connected.

If T'' is *not* returned at step 3, we have $c_{root} \in C$. It is important to observe that in this case, the roots of the out-trees in T' at step 2 are exactly the children of c_{root} in T. This is because the root of an out-tree in F has an incoming arc in T' if and only if its parent in T is of type (b).

Secondly suppose that T'' is returned at step 4. Then c_{root} does not participate in T' and c_{root} in T'' is of in-degree 0. By the observation in the second paragraph, T'' is an out-tree.

Thirdly suppose that T'' is returned at step 5. In this case c_{root} has been included as an internal vertex to connect two out-trees in step 1, and the arcs (f_p, c_{root}) and (c_{root}, f_q) have been included in T', where $f_p f_q$ is the pair vertex found in step 5.1. We want to check that c_x and the arc (c_x, x) in line 5.3 can be properly picked up. Indeed, the pair vertex $f_p x$ belongs to $H_{root} \subseteq H$ and there exists a vertex c_x which is matched to the pair $f_p x$. By the construction of B, the arc (c_x, x) exists as well. Hence at the end of step 5.4, T'' is a union of out-trees whose roots are c_x and the children of c_{root} in T other than x.

If $d^+_T(c_{root}) = 1$, T'' consists of a single out-tree whose root is c_x. Else if $d^+_T(c_{root}) \geq 2$, let y be a child of c_{root} in T and $y \neq x$. Since $(f_p, c_{root}), (c_{root}, y) \in A(D)$, we have the pair vertex $f_p y$ in $H_{root} \subseteq H$ and $f_p y$ is uniquely matched to a vertex c_y. The edge $\{f_p y, c_y\}$ implies the existence of the two arcs (f_p, c_y), (c_y, y), hence we can perform step 5.5 properly. Since the vertex f_p is contained in the out-tree rooted at $c_x \in C_m$, the addition of these arcs does not create a cycle. As a result we start at the step 5.5 with $|d^+_T(c_{root})|$ out-trees in the beginning and each time we carry out step 5.5.2, the number of out-trees in T'' decreases by 1. Therefore at the end of step 5.5, we end up with a single out-tree T'' rooted at c_x. □

During the construction of T'', we added at least one vertex c_{pq} for each internal vertex c_i of type (b) as an internal vertex of T''. Also we added at least one vertex as the root or an internal vertex of T'' if $c_{root} \in C$. Hence the number of internal vertices in C for T'' is at least as large as the number of internal vertices in C for T. Therefore what remains is to see that every vertex f of F which is internal in T can be made to remain internal. The only case we need to consider is a vertex $f \in V(F)$ whose children in T are leaves and all belong to C. Suppose f is a leaf in T'. Since $f \in N(C) = H$, we can uniquely determine a vertex $c_f \in C_m$ such that $\{f, c_f\}$ belongs to the perfect matching M. By the construction of T'' in the above argument, the vertex c_f is not contained in T'' for each such vertex $f \in V(F)$ and thus, we may add c_f and an arc (f, c_f) to T'' while keeping T'' as an out-tree. After this procedure each such vertex f is an internal vertex in T'', and thus T'' has as many internal vertices as T.

For any vertex c of C_m which does not participate in T' constructed so far, we simply add it to T'' with the arc $(f, c) \in A(D)$. Therefore T'' is an out-branching of $D - C_u$ with as many internal vertices as T. This completes the proof. □

In light of Lemma 5.4.4, we have a reduction rule below.

Reduction Rule 1. *Given a digraph D with a vertex cover U of D and $W = V(D) \setminus U$, construct the associated bipartite graph B. If B has a crown $(H, C = C_m \cup C_u)$ with $C_u \neq \emptyset$, remove the vertices of C_u from D.*

We need the following theorem to prove our kernelization lemma.

Theorem 5.4.5. *[52] Any graph G with an independent set I, where $|I| \geq \frac{2n}{3}$, has a crown (H, C), where $H \subseteq N(I)$, $C \subseteq I$ and $C_u \neq \emptyset$, that can be found in time $O(nm)$ given I.*

Lemma 5.4.6 (Kernelization Lemma)**.** *Let D be irreducible. If $|V(D)| > 8k^2 + 6k$ then D has an out-branching with at least k internal vertices.*

Proof. Suppose that D is reduced with $|V(D)| > 8k^2 + 6k$, and that D does not have an out-branching with at least k internal vertices. Since the internal number of D is the same as the internal number of the original digraph, we may assume that D has an out-branching T.

For $|U| < 2k$, we have $|W| = |V(D) \setminus U| > 8k^2 + 4k$ and $|U'| = |N^{-}(W) \cup (U \times U)| < 2k + 4k^2$. Then $|W| \geq \frac{2|V(B)|}{3}$ which means we have a crown $(H, C = C_m \cup C_u)$ of D with $C \subseteq W$ and $C_u \neq \emptyset$ by Theorem 5.4.5. This is a contradiction to that D is reduced. □

Proceeding from what has been discussed above, we give a polynomial time algorithm which computes a quadratic order kernel for Directed k-Internal.

KERNELIZATION

1. Build an out-branching T rooted at r by depth-first search.
2. Transform T into a minimal out-branching using 1-change.
3. **If** the number of leaves of T is at most $n - k$, return 'YES'.
4. **Otherwise** Reduce by Rule 1 if possible. If this is not possible, return the instance (it is irreducible).
   Let T be the new out-branching obtained by the construction in the proof of Lemma 5.4.4.
   Transform T into a minimal out-branching using 1-change.
   Go to line 3.

Step 1-3 take $O(n^2 m)$ time by Lemma 5.4.2. At step 4, we can construct the bipartite graph B in time $O(n^3)$, and $V(B)$ and $E(B)$ are bounded by $n + 2k + 4k^2 = O(n^2)$ and

$m + 4k^2n = O(n^3)$ respectively. Due to Theorem 5.4.5, in $O(n^5)$ time we can reduce the instance by Rule 1 or declare the instance irreducible. Since the size of an instance is strictly decreased at each step of the reduction, we conclude that the algorithm KERNELIZATION runs in $O(n^6)$ time.

5.5 FPT-algorithm for Directed k-Internal

The quadratic kernel in the previous section suggests a trivial fpt-algorithm based on exhaustive search. In order to achieve a better running time we provide an alternative way of showing the fixed-parameter tractability of the Directed k-Internal problem based on the notion of *tree decomposition*.

Theorem 5.5.1. *There is a polynomial time algorithm that, given an instance (D, k) of the* Directed k-Internal *problem, either finds a solution or establishes a tree decomposition of D of width at most $2k - 2$.*

Proof. By Lemma 5.4.2, there is a polynomial time algorithm which either finds a solution or specifies a vertex cover C of D of size at most $2k - 2$. Let $I = \{v_1, \dots, v_s\} = V(D) \setminus C$. Consider a star U with nodes $x_0, x_1, \dots, x_s$ and edges $x_0x_1, x_0x_2, \dots, x_0x_s$. Let $X_0 = C$ and $X_i = X_0 \cup \{v_i\}$ for $i = 1, 2, \dots, s$ and let X_j be the bag corresponding to x_j for every $j = 0, 1, \dots, s$. Observe that $(\{X_0, X_1, \dots, X_s\}, U)$ is a tree decomposition of D and its width is at most $2k - 2$. □

Theorem 5.5.1 shows that an instance (D, k) of the Directed k-Internal can be reduced to another instance with treewidth $O(k)$. Using standard dynamic programming techniques we can solve this instance in time $2^{O(k \log k)} n^{O(1)}$ [3, 28].

We can further accelerate the solution procedure using kernelization. If we first find the kernel and then establish the tree decomposition, the resulting algorithm will run in time $O(2^{O(k \log k)} + n^6)$. Now we have the following result.

Theorem 5.5.2. *The* Directed k-Internal *problem can be solved by an additive FPT algorithm of running time $O(2^{O(k \log k)} + n^6)$.*

5.5.1 Dynamic Programming on Graphs with Bounded Treewidth

In this subsection, we give a description[1] of the dynamic programming for MinLeaf on (directed) graphs of treewidth w of running time $O(2^{O(w \log w)})$. We assume that the given

[1] The standard dynamic programming is used for the maximum leaf problem [3, 28] on graph with bounded treewidth. Up to the best of the author's knowledge, however, no description of the dynamic programming can be found in known literatures. We also acknowledge the early discussion with Paul Bonsma on this topic.

tree decomposition is nice, see the notion in Section 2.1.

In the following exposition, we follow the notion of [22]. Let (X, T) be a nice tree decomposition of D of width w. We reserve n_i to denote the number of elements in the bag X_i. For a node $i \in V(T)$, let $D_i = (V_i, A_i, X_i)$ be the subdigraph of D. The vertex set is $V_i = \bigcup_{j \in V(T_i)} X_j$, where T_i be the subtree of T rooted at i, and the arc set A_i contains all arcs whose both endpoints lie inside V_i. Here are more notions we shall use in our exposition.

- A *solution* of MinLeaf is an out-branching in D.
- A *partial solution* F on V_i is a spanning *out-forest* on V_i which can be extended to an out-branching in D.
- An *extension* F' on V' of a partial solution F on V is a spanning out-forest such that $F'[V] = F$, i.e., a spanning out-forest of V' from which the deletion of $V' - V$ vertices leads to F. F is said to be a *restriction* of F' on V.
- The *configuration* of a partial solution F on V_i is the information on F we need in order to decide it can be extended into a solution.

  Suppose we're dealing with Introduce Node i and its descendant j. For MinLeaf, given a new vertex $x \notin V_j$, we need to know the followings: (1) in case x is to be attached to $y \in V_j$ as a *parent*, y should be a *root* in a partial solution F on V_j. (2) in case x is to be attached as a *child*, any vertex $y \in V_j$ can be its parent, but we choose exactly one. (3) in case x is attached as a parent to y (or more such vertices) and as a child to y', we need to make sure that y and y' belong to *different components* of f. Indeed we need to know which vertices are leaves in order to keep track of them. In summary, the information we need about F is, whether $F \in V_j$ is a root or not, a leaf or not, and which component it belongs to. It turns out that these are also sufficient to perform our dynamic programming.

To simplify the description of our dynamic programming algorithm, we will consider the alternative problem of rooted MinLeaf, in which we want to find an out-branching with minimum number of leaves rooted at a specified vertex r. Once we prescribe the root r, it is not difficult to rearrange the given nice tree decomposition (X, T) so that the root node of T contains r. We assume this without loss of generality. This assumption is useful due to the following lemma.

Lemma 5.5.3. *Let (X, T) be a nice tree decomposition such that r is contained in the bag of the root node of T. For any out-branching F of D, its restriction $F[V_i]$ on V_i has all the roots in X_i.*

Proof. Observe that the restriction $F[V_i]$ is obtained by deleting the vertices in $V(D) - V_i$. Since X_i is a cut and its removal from D disconnects $D[V_i - X_i]$ and $V(D) - V_i$, there is no incoming arc from $V(D) - V_i$ to $V_i - X_i$. Now the claim follows. □

For a partial solution F_i on V_i and for each vertex $x \in X_i$, the information we need are the followings (1) whether or not x is a root in F_i, (2) whether or not x is a leaf, (3) the root of the component of F_i to which x belongs. By Lemma 5.5.3, the vertex of (3) always appear in X_i. Therefore, for each bag X_i, the size of the table does not exceed $(2 \cdot 2 \cdot n_i)^{n_i} \leq (4w)^w$. Let f^{root}, f^{leaf} and f^{comp} denote the mappings for (1),(2) and (3) respectively. That is:

1. $f^{root} : X_i \to \{0, 1\}$. We set $f^{root}(x) = 1$ if and only if x is a root in the corresponding partial solution F_i on V_i.

2. $f^{leaf} : X_i \to \{0, 1\}$. We set $f^{leaf}(x) = 1$ if and only if x is a leaf in the corresponding partial solution F_i on V_i.

3. $f^{comp} : X_i \to X_i$. We set $f^{comp}(x) = z$ if z is the root of the component x belongs to.

We call a vector $f \in \{\{0, 1\} \times \{0, 1\} \times X_i\}^{n_i}$ a *configuration* and the first, second and third field of $f(x)$ each represents $f^{root}(x)$, $f^{leaf}(x)$ and $f^{comp}(x)$. For a given partial solution F, a vector $f \in \{\{0, 1\} \times \{0, 1\} \times X_i\}^{n_i}$ is said to be the configuration of F, denoted as $c(F)$, if it satisfies the above three conditions. We may consider a restriction of a configuration on a subset S and in this case we say $c(F)$ is the configuration of F on S. For two configurations f', f'' on disjoint sets S', S'' respectively, the concatenation of f' and f'' is denoted $f' \oplus f''$.

For a configuration f, the value $\ell(f)$ denotes the minimum number of leaves in a partial solution F_i on V_i whose configuration is f. The number $\#_{leaf}(f) := |\{x \in X_i : f^{leaf}(x) = 1\}|$ denotes the number of vertices in X_i which are assigned as a leaf in the configuration f. Notice that it is possible that some configuration f is *infeasible* in the sense that no partial solution exists with such a configuration. If f is an infeasible configuration, we set $\ell(f) := \infty$.

Some obvious conditions for feasibility are (a) if $f^{root}(x) = 1$, then $f^{comp}(x) = x$, (b) if $f^{comp}(x) = z$ then $f^{root}(z) = 1$. Whether f satisfies the conditions (a) and (b) can checked easily and we assume that the dynamic programming do this as a preprocessing in every computation of the table as a part of preprocessing. Now we present the detail of the dynamic programming. Regarding the running time of each step, we hide the polynomial factor.

Leaf Node

For each leaf node i of T, we initialize the table for X_i by enumerating all possible configurations and checking the feasibility of each configuration. For each configuration f for X_i, We set:

$$\ell(f) := \begin{cases} |(f^{leaf})^{-1}(1)| & \text{if } f \text{ is feasible} \\ \infty & \text{otherwise} \end{cases}$$

In order check the feasibility of a configuration f, we execute a simple procedure. First, we delete all outgoing from the vertex set $(f^{leaf})^{-1}(1)$ and all incoming arcs into the vertex set $(f^{root})^{-1}(1)$. As the mapping f^{comp} partitions X_i into predeterminate components, we consider each component and examine whether there is an out-branching with the desired leaves and the root. Let $S(z) = \{x \in X_i : f^{comp} = z\}$ be the set of vertices in X_i which consists the component rooted at z. With $D[S(z)]$, we perform the polynomial-time algorithm introduced in Section 5.1 for every acyclic ordering of $S(z)$. If there is an out-branching $F(z)$ of $S(z)$ for some acyclic ordering with $leaf(F(z)) = (f^{leaf})^{-1}(1) \cap S(z)$, and if this is true for every component $S(z)$ induced by f, then obviously f is a feasible configuration. Conversely, we can find an out-branching $F(z)$ for each component $S(z)$ for some acyclic ordering if f is a feasible configuration. We call this procedure FeasibilityCheck. This procedure will be used to update the table at a join node.

For each configuration f, computing the value $\ell(f)$ amounts to checking the feasibility of f. Observe that the running time of FeasibilityCheck is $O(w!) \approx O(2^{O(w \log w)})$.

Forget Node

Let i be a forget node, j be its child and $X_i = X_j - \{x\}$. For each configuration $f_i \in \{\{0,1\} \times \{0,1\} \times X_i\}^{n_i}$, we set

$$\ell(f_i) := \min_{\text{all configurations } f \text{ for } X_j} \{\ell(f) : f_i = f|_{X_i} \text{ and } f^{root}(x) = 0\}.$$

Put in another way, we consider every configuration f whose restriction on X_i is f_i and in which x is not appointed as a root. By Lemma 5.5.3, no partial solution F with $c(F) = f$ can be extended into a full partial solution if $f^{root}(x) = 0$. This is why we only consider f with $f^{root}(x) = 0$.

For each f_i, comparing with the configurations f in the table X_j takes $O(n_j)$ time and the total running time to build the table for X_i is $O(n_i \cdot (4w)^w) \approx O(n_i \cdot 2^{O(w \log w)})$. It is clear that $\ell(f_i)$ is finite if and only if f is a feasible configuration, and the value $\ell(f)$ correctly represent the number of leaves in a partial solution on V_i with the configuration f.

Introduce Node

Let i be an introduce node, j be its child and $X_i = X_j \cup \{x\}$. Then for each configuration $f_i \in \{\{0,1\} \times \{0,1\} \times X_i\}^{n_i}$, we set the value $\ell(f_i)$ as follows.

If $f_i^{root}(x) = 1$, we set:

$$\ell(f_i) := \begin{cases} \min \ell(f) + 1 & \text{if } f_i^{leaf}(x) = 1 \\ \min \ell(f) & \text{otherwise} \end{cases}$$

If $f_i^{root}(x) = 1$ and $f_i^{comp}(x) = z$ for some $z \in X_j$, we set:

$$\ell(f_i) := \begin{cases} \infty & \text{if } \nexists y \in N^-(x) \text{ with } f_i^{comp}(y) = z \\ \min \ell(f) + 1 & \text{if } \exists y \in N^-(x) \text{ with } f_i^{comp}(y) = z,\, f_i^{leaf}(y) = 0 \\ \min \ell(f) & \text{if } f_i^{leaf}(y) = 1,\, \forall y \in N^-(x) \text{ with } f_i^{comp}(y) = z \end{cases}$$

Here the minimum is taken over all configurations f such that

$$f = f_i|_{X_j}$$

in case of $f_i^{leaf}(x) = 1$. In case of $f_i^{leaf}(x) = 0$, the minimum is taken over all configurations f such that

$$\begin{array}{ll} \forall y \notin Y & f(y) = f_i(y) \\ \forall y \in Y & \begin{cases} f^{leaf}(y) = f_i^{leaf}(y) \\ f^{comp}(y) \in (N^+(x) \cap Y)(\cup\{z\} \text{ if } f_i^{comp}(x) = z) \end{cases} \end{array}$$

where $Y = \{y \in X_j : f_i^{comp}(y) = f_i^{comp}(x)\}$.

Considering the four possible root/leaf configurations of x, it is not difficult to check that the above assignments are correct and the computation requires $O(2^{O(w \log w})$ time.

Join Node

Let i be a join node, j and k be its children with $X_i = X_j = X_k$. For each configuration $f \in \{\{0,1\} \times \{0,1\} \times X_i\}^{n_i}$, we want to compute the value $\ell(f)$ using the tables for X_j and X_k. Give a partial solution F on V_i, we observe that

$$|leaf(F)| = |leaf(F_j)| + |leaf(F_k)| - \#_{leaf}(c(F_j)) - \#_{leaf}(c(F_k)) + \#_{leaf}(c(F) \qquad (5.2)$$

where $F_j = F[V_j]$ and $F_k = F[V_k]$. In other words, the set $leaf(F_i)$ is the disjoint union of the sets $leaf(F) \cap (V_j - X_i)$, $leaf(F) \cap X_i$ and $leaf(F) \cap (V_k - X_i)$.

Take a configuration f for X_i. By definition, we have

$$\ell(f) := \begin{cases} \min_{\text{all partial solutions } F \text{ with } c(F) = f} |leaf(F)| & \text{if } f \text{ is feasible} \\ \infty & \text{otherwise.} \end{cases} \tag{5.3}$$

We can evaluate $\ell(f)$ using the values $\ell(f_j)$, $\ell(f_k)$ in the already processed tables and avoid the hassle of going over all partial solutions of arbitrary size. For this, we introduce a notion of *f-consistent pair*. Let f_j and f_k be a configuration for X_j and X_k each. Roughly speaking, a pair (f_j, f_k) is f-consistent if any partial solutions F_j and F_k with $c(F_j) = f_j$, $c(F_k) = f_k$ can be combined into a partial solution F with $c(F) = f$, possibly after adding some arcs in X_i. We defer the definition of f-consistent pair. Once we present the definition, we will also give the algorithm ConPairs which generates all f-consistent pairs. The update of the table for X_i is based on the following equation.

$$\ell(f) = \begin{cases} \min_{\text{all } f\text{-consistent pairs } (f_j, f_k)} & \ell(f_j) + \ell(f_k) - \#_{leaf}(f_j) - \#_{leaf}(f_k) + \#_{leaf}(f) \\ & \text{if } f \text{ is feasible} \\ \infty & \text{otherwise.} \end{cases} \tag{5.4}$$

Let F be a partial solution on V_i. Every vertex of F belongs to exactly one of the three parts, i.e. $V_j - X_i$, $V_k - X_i$ and X_i. Let $Y_j = V_j - X_i$ and $Y_k = V_k - X_i$. We say a vertex x of X_i is a *boundary vertex* if it is adjacent with a vertex of Y_j or Y_k, and let B be the set of all boundary vertices. From F, we obtain a spanning out-forest F', which we call the *B-trim of F*, by deleting all arcs between X_i and B. Notice that they include all arcs between vertices of B. Here is the definition of f-consistent pair.

- Let F be a partial solution on V_i, F' be the B-trim of F. We take $F_j := F'[V_j]$ and $F_k := F'[V_k]$. The two configurations $f_j := c(F_j)$ and $f_k := c(F_k)$ are called an *F-consistent pair.*

- Given a configuration f for X_i, we say two configurations f_j and f_k (for X_j and X_k respectively) are an *f-consistent pair* if it is an F-consistent pair for some a partial solution F on V_i with $c(F) = f$.

Notice that by definition, the F-consistent pair for a given partial solution F is uniquely decided. Also, there exists an f-consistent pair if and only if f is a feasible configuration. Observe that we can reconstruct the original partial solution F from $F'[V_j]$ and $F'[V_k]$, where F' is the F-trim of F, by adding the deleted arc set $F - F'$. In fact, if we replace $F'[V_j]$ and $F'[V_k]$ by different partial solutions with the same configurations, we can obtain a partial solution on V_i by adding the arc set $F - F'$. Notice that we construe an out-forest as a set of arcs.

Lemma 5.5.4. *Let F be a partial solution on V_i, F' be the B-trim of F, and f_j and f_k be the F-consistent pair. Then, for any partial solution F_j on V_j and F_k on V_k with $c(F_j) = f_j$, $c(F_k) = f_k$, the graph F_i obtained as*

$$F_j[Y_j \cup B] \cup F_k[Y_k \cup B] \cup (\textit{exactly one of } F_j[X_i - B] \textit{ and } F_k[X_i - B]) \cup (F - F')$$

is a partial solution on V_i whose configuration on X_i is exactly f.

Proof. It is sufficient to check that (a) every vertex x of V_i has exactly on incoming arc in F_i, (b) F_i is acyclic, and (c) the set of roots in F_i is exactly $(f^{root})^{-1}(1)$ and the set of leaves contained X_i, i.e. $leaf(F_i) \cap X_i$, is exactly $(f^{leaf})^{-1}(1)$.

Let F'' be the subgraph $F_j[Y_j \cup B] \cup F_k[Y_k \cup B] \cup$ (exactly one of $F_j[X_i - B]$), to which we did not add the set $F' - F$ yet. We shall show that (a), (b) is true for F''. Then we show that adding the arc set does not violate (a),(b) and the resulting graph F_i satisfies (c).

For every vertex $x \in Y_j \cup Y_k$ and $x \in X_i - B$, the condition (a) is obviously true. Consider a vertex $x \in B$ and suppose it has an incoming arc in each of $F_j[Y_j \cup B]$ and $F_k[Y_k \cup B]$. It is implied that x has an incoming arc in each of F_j and F_k, and thus we have $f_j^{root}(x) = f_k^{root}(x) = 0$. Since we define $f_j := c(F'[V_j])$ and $f_k := c(F'[V_k])$, this implies that x has an incoming arc in each of $F'[V_j]$ and $F'[V_k]$. Recalling that there is no arc between vertices of B, the incoming arcs into x should be distinct in $F'[V_j]$ and $F'[V_k]$, contradiction to the fact that F' is an out-forest. Therefore, (a) is true for $x \in B$.

To see the condition (b), suppose there exists a directed cycle in F''. Since there is no arc between $X_i - B$ and B in F'', the only possibility that a cycle may occur is in the graph $F_j[Y_j \cup B] \cup F_k[Y_k \cup B]$. However, due to the fact that $c(F_j) = f_j = c(F'[V_j])$ and $c(F_k) = f_k = c(F'[V_k])$, the existence of a cycle in $F_j[Y_j \cup B] \cup F_k[Y_k \cup B]$ implies the existence of a cycle in $F'[Y_j \cup B] \cup F'[Y_k \cup B]$, a contradiction.

Now we consider the graph $F_i := F'' \cup (F - F')$. Without loss of generality we choose $F_j[X_i - B]$. Observe that for any arc $xy \in F - F'$, y is a root in F' and thus $f_j^{root}(y) = f_k^{root}(y) = 1$. Therefore the condition (a) is not violated by the addition of $F - F'$. The condition (b) is valid as well. Indeed, $c(F_j) = c(F'[v_j])$ and $c(F_k[Y_k \cup B]) = F'[Y_k \cup B])$. Observing that $F'' = F_j \uplus F_k[Y_k \cup B]$, $F' = F'[V_j] \cup F'[Y_k \cup B]$ and $F_i - F'' = F - F'$, the existence of a cycle in F_i implies the existence of a cycle in F, which is impossible.

Lastly we look at the condition (c). As F'' and F' has the same set of roots and leaves in X_i, it remains to observe the obvious fact that adding $F - F'$ to F' satisfies the condition (c).

□

Concerning the validity of the assignment (5.4), we have the following lemma.

Lemma 5.5.5. *The assignments of* (5.4) *is correct.*

Proof. We claim that the right-hand side of (5.4) gives a lower bound on $\ell(f)$. If f is not feasible, this is trivially true, hence we assume that f is feasible. Take a partial solution F with $c(F) = f$ attaining the minimum $\ell(f)$ in (5.3), and replace $|leaf(F)|$ by the equation (5.2). Let F' be the B-trim of F. Since $f_j := c(F'[V_j])$ and $f_k := c(F'[V_k])$ is an F-consistent pair, and thus an f-consistent pair, it follows that the right-hand side of (5.4) does not exceed $\ell(f)$.

For the other direction, suppose f_j and f_k is a f-consistent pair attaining the minimum in the right-hand side of (5.4). By definition, they form an F-consistent pair for some partial solution F on V_i and let F' be the B-trim of F. We take partial solutions F_j on V_j and F_k on V_k with $c(F_j) = f_j$ and $c(F_k) = f_k$, which attains the minimum $\ell(f_j)$ and $\ell(f_k)$ respectively. Let F_i be a partial solution obtained from F_j and F_k in the way described in Lemma 5.5.4. Then Equation (5.2) means that the right-hand side of (5.4) counts the number of leaves in a partial solution F_i and bviously $\ell(f)$ provides a lower bound on this value. This establishes the equation (5.4). □

The remaining question is how to generate all f-consistent pairs. The observation in the following lemma is crucial to design the algorithm ConPairs.

Lemma 5.5.6. *Let F' be the B-trim of a partial solution F on V_i. We take $F_s := F'[X_i - B]$, $F^*_j := F'[Y_j \cup B]$ and $F^*_k := F'[Y_k \cup B]$ and let f_s, f^*_j and f^*_k respectively be the configurations of F_s on $X_i - B$, F^*_j on B and F^*_k on B. Then the F-consistent pair f_j, f_k is given by $f_j = f_s \oplus f^*_j$ and $f_k = f_s \oplus f^*_k$.*

Proof. By definition of F-consistent pair, $f_j = c(F'[V_j])$ and $f_k = c(F'[V_k])$. As B-trim deletes all arcs between B and X_i, we have $F'[V_j] = F_s \uplus F^*_j$ and $F'[V_k] = F_s \uplus F^*_k$, where $\uplus$ denotes the disjoint union operation. The claim follows. □

With Lemma 5.5.6, the first step to generate f-consistent pair is to guess the boundary vertex set B, and then guess f_s on $X_i - B$, and f^*_j,f^*_k on B. When we do this, any two vertices should be assigned to distinct components in these these configurations whenever they belong to distinct components in f. Moreover, a vertex should be assigned as a root (respectively, leaf) whenever it is assigned as a root (respectively, leaf) in f.

- Given a configuration f for X_i, we say f' is an *f-consistent configuration* if, for every vertex $x \in X_i$, (a) $f'^{root}(x) = 1$ whenever $f^{root}(x) = 1$, (b) $f'^{leaf}(x) = 1$ whenever $f^{leaf}(x) = 1$, and for every $x, y \in X_i$ (c) $f'^{comp}(x) \neq f'^{comp}(y)$ whenever $f^{comp}(x) \neq f^{comp}(y)$.

In addition, f^*_j and f^*_k on B should satisfy $(f^*_j)^{root}(x) = 1$ or $(f^*_k)^{root}(x) = 1$ for every $x \in B$ since otherwise x has two incoming arcs. Also f_s should be feasible w.r.t $X_i - B$. In other words, the configuration f_s should represent a spanning out-branching *on* $X_i - B$ since f_s represents the components of the B-trim in $X_i - B$ after deleting arcs adjacent with vertices in B. The feasibility of f_s can be checked using the procedure FeasibilityCheck described in Leaf Node part. Another important condition is that no directed cycle structure is induced by the triple configurations. To clarify this point, let us introduce an auxiliary graph P.

We build a *pattern graph* P using the information of the given (f, B)-distillation triple (f_s, f_j^*, f_k^*). At the beginning, P consists of the vertices X_i, which are independent. For each of f_s, f_j^*, f_k^*, we do the following. For every component in the configuration, we make a directed path which starts from the root of the component and traversing every non-leaf vertices of the component. The leaves of the components is attached to the last vertex of the path.

We summarize the conditions of the triple discussed so far.

- Given a configuration f for X_i, let B be a subset of X_i. The configurations f_s on $X_i - B$, f_j^* and f_k^* on B is called an (f, B)-*distillation* triple if they satisfy the followings.

  1. Each of them are f-consistent.
  2. f_s is feasible w.r.t $X_i - B$.
  3. $(f_j^*)^{root}(x) = 1$ or $(f_k^*)^{root}(x) = 1$ for every $x \in B$.
  4. The pattern graph P does not have a cycle.

It is not difficult to see that if (f_j,f_k) is an f-consistent pair, then $f_s := f_j|_{X_i - B}$, $f_j^* := f_j|_B$ and $f_k^* := f_k|_B$ form an (f, B)-distillation triple. Hence, by examining all (f, B)-distillation triple for every subset B, we do not miss any f-consistent pair. Not all of them, however, lead to an f-consistent pair. In order to check whether a given pair $f_j = f_s \oplus f_j^*$, $f_k = f_s \oplus f_k^*$ is consistent, we add a set R of arcs whose one endpoint belongs to B. More specifically, we say R is complementary if

1. every arc lies between $X_i - B$ and B,
2. the set of all head vertices in R is exactly the union of $(f_s^{root})^{-1}(1) - (f^{root})^{-1}(1)$ and $[(f_j^{root})^{-1}(1) \cap (f_k^{root})^{-1}(1)] - (f^{root})^{-1}(1)$ without duplication, and
3. R contains at least one outgoing arc from x for every $x \in (f_s^{leaf})^{-1}(1) - (f^{leaf})^{-1}(1)$ and every $x \in [(f_j^{leaf})^{-1}(1) \cap (f_k^{leaf})^{-1}(1)] - (f^{leaf})^{-1}(1)$.
4. R does not induce a directed cycle when added to the pattern graph P.

Algorithm 8 Compute the value $\ell(f)$.

```
Require: A configuration $f$ for $X_i$.
Ensure: Output $\ell(f)$.
Initialize $\ell(f) := \infty$
for all $B \subseteq X_i$ do
    for all $(f, B)$-distillation triple $(f_s, f_j^*, f_k^*)$ do
        for all complementary subsets $R$ do
            $f_j := f_s \oplus f_j^*$ and $f_k := f_s \oplus f_k^*$
            $\ell(f) := \min\{\ell(f), \ell(f_j) + \ell(f_k) - \#_{leaf}(f_j) - \#_{leaf}(f_k) + \#_{leaf}(f)\}$
        end for
    end for
end for
if no pair was output then return "NO"
```

Now we are ready to present the algorithm ConPairs. It is not difficult to check that the running time of the algorithm is $O(2^{O(w \log w)})$.

The correctness of ConPairs follows from the following lemma.

Lemma 5.5.7. *Let f be a configuration for X_i. Then f is feasible if and only if there exists a subset $B \subseteq X_i$, an (f, B)-distillation triple and a complementary set R w.r.t to the triple such that each of $f_j := f_s \oplus f_j^*$ and $f_k := f_s \oplus f_k^*$ is feasible.*

Proof. The sufficiency of the condition is easy to see. For feasible f, there exists a partial solution F with $c(F) = f$. Take the boundary vertices as B, construct the triple (f_s, f_j^*, f_k^*) as explained in Lemma 5.5.6 and let $R := F - F'$, where F' is the B-trim. Clearly the triple is (f, B)-distillation and R is complementary.

For the other direction, we first note that $P + R$, the graph on X_i obtained by adding R to P has the configuration f. Since f_j and f_k are feasible, there are partial solutions F_j on V_j and F_k on V_k. Let F be a graph on V_i obtained as

$$F_j[Y_j \cup B] \cup F_k[Y_k \cup B] \cup (\text{exactly one of } F_j[X_i - B] \text{ and } F_k[X_i - B]) \cup R.$$

It remains to observe that F is a partial solution on V_i with $c(F) = f$. □

5.6 Improved FPT-algorithm for Directed k-Internal

In this section we describe an fpt-algorithm for Directed k-Internal which repeatedly solves the k-Out-Tree problem introduced in Chapter 4. The basic idea behind the new algorithm is that we can embed an out-tree with k internal vertices whenever there is an out-branching with k internal vertices and that we can effectively bound the number of vertices in such an embedded out-tree by a function of k. Now we produce all possible forms of embedded out-trees of bounded size and check whether the given digraph contains a fixed out-tree for each of them.

A *k-internal out-tree* is an out-tree with at least k internal vertices. We call a k-internal out-tree *minimal* if none of its proper subtrees is a k-internal out-tree, or *minimal k-tree* in short.

Lemma 5.6.1. *Let T be a k-internal out-tree. Then T is minimal if and only if $|Int(T)| = k$ and every leaf $u \in Leaf(T)$ is the only child of its parent $N^-(u)$.*

Proof. Assume that T is minimal. It cannot have more than k internal vertices, because otherwise by removing any of its leaves, we obtain a subtree of T with at least k internal vertices. Thus $|\mathrm{Int}(T)| = k$. If there are sibling leaves u and w, then removing one of them provides a subtree of T with $|\mathrm{Int}(T)|$ internal vertices.

Now, assume that $|\mathrm{Int}(T)| = k$ and every leaf $u \in \mathrm{Leaf}(T)$ is the only child of its parent $N^-(u)$. Observe that every subtree of T can be obtained from T by deleting a leaf of T, a leaf in the resulting out-tree, etc. However, removing any leaf v from T decreases the number of internal vertices, and thus creates subtrees with at most $k - 1$ internal vertices. Thus, T is minimal. □

By definition a minimal k-tree is also a k-internal out-tree. Due to the following simple observation, a digraph D has a k-internal out-tree rooted at r if and only if it has a minimal k-tree rooted at r.

Lemma 5.6.2. *Any k-internal out-tree rooted at r contains a minimal k-tree rooted at r as a subdigraph.*

Moreover, combined with Lemma 2.1.1, the next lemma attributed to folklore shows that in order to see whether there is an out-branching with at least k internal vertices it suffices to check the existence of a minimal k-tree rooted at r for all $r \in S$. Here S is the unique strong connectivity component of D without incoming arcs. Recall that by Lemma 2.1.1 more than one strong component of D without incoming arcs implies there is no out-branching at all.

Lemma 5.6.3. *Suppose a given digraph D has an out-branching rooted at vertex r. Then any minimal k-tree rooted at r can be extended to a k-internal out-branching rooted at r in time $O(m + n)$.*

So far we have reduced the problem Directed k-Internal into the problem of finding a minimal k-tree rooted at r. The latter problem is essentially as difficult as the former one, but fortunately the size of any minimal k-tree is effectively bounded by a small function of k as we shall see. In fact, Lemma 5.6.1 can be used to generate all non-isomorphic minimal k-trees. First, build an (arbitrary) out-tree T^0 with k vertices. Then extend T^0 by adding a vertex x' for each leaf $x \in \text{Leaf}(T^0)$ with an arc (x, x'). The resulting out-tree T' satisfies the properties of Lemma 5.6.1. Conversely, by Lemma 5.6.1, any minimal k-tree can be constructed in this way. A minimal k-tree has the maximum number of vertices when T^0 is a star, and this number is $2k - 1$.

Generating Minimal k-Tree (GMT) Procedure

a. Generate a k-vertex out-tree T^0 and a set $T' := T^0$.

b. For each leaf $x \in \text{Leaf}(T')$, add a new vertex x' and an arc (x, x') to T'.

Our algorithm for Directed k-Internal on a digraph D runs in two stages. In the first stage, we generate *all* minimal k-trees. We use the GMT procedure described above to achieve this. At the second stage, for each $r \in S$ and each minimal k-tree T, we check whether D contains an out-tree rooted at r and isomorphic to T using the algorithm from Chapter 4. We return TRUE if and only if we succeed in finding an out-tree H of D rooted at $r \in S$ which is isomorphic to a minimal k-tree.

In the literature, mainly rooted (undirected) trees and not out-trees are studied. However, every rooted tree can be made an out-tree by orienting every edge away from the root and every out-tree can be made a rooted tree by disregarding all orientations. Thus, rooted trees and out-trees are equivalent and we can use results obtained for rooted trees for out-trees.

Otter [100] showed that the number of non-isomorphic out-trees on k vertices is $t_k = O^*(2.95^k)$. We can generate all non-isomorphic rooted trees on k vertices using the algorithm of Beyer and Hedetniemi [17] of runtime $O(t_k)$. Using the GMT procedure we generate all minimal k-trees. We see that the first stage of the algorithm can be completed in time $O^*(2.95^k)$.

In the second stage, we try to find a copy of a minimal k-tree T in D using the deterministic algorithm for k-Out-Tree from Chapter 4. The running time of this subroutine is $O^*(6.14^k)$. Since the number of vertices of T is bounded from above by $2k-1$, the overall running time for the second stage of the algorithm is $O^*(2.95^k \cdot 6.14^{2k-1})$. Thus, the overall time complexity of the algorithm is $O^*(2.95^k \cdot 6.14^{2k-1}) = O^*(112^k)$.

We can reduce the complexity with a more refined analysis of the algorithm. The major contribution to the large constant 112 in the above simple analysis comes from the running time of the iterative subroutines executed to find an isomorphic copy of a minimal k-tree. There we use the upper bound on the number of vertices in a minimal k-tree. However, most of the minimal k-trees have less than $k - 1$ leaves, which implies that the upper bound $2k - 1$ on the order of a minimal k-tree is too big for the majority of the minimal k-trees. Let $T(k)$ be the running time. Then we have

$$T(k) = O^*\left(\sum_{k+1\leq k'\leq 2k-1} (\text{\# of minimal } k\text{-trees on } k' \text{ vertices}) \times (6.14^{k'})\right) \tag{5.5}$$

A minimal k-tree T' on k' vertices has $k' - k$ leaves, and thus the out-tree T^0 from which T' is constructed has k vertices of which $k' - k$ are leaves. Hence the number of minimal k-trees on k' vertices is the same as the number of non-isomorphic out-trees on k vertices with $k' - k$ leaves. Here an interesting counting problem arises. Let $g(k, l)$ be the number of non-isomorphic out-trees on k vertices with l leaves. Enumerate $g(k, l)$. To our knowledge, such a function has not been studied yet. Leaving it as a challenging open question, here we give an upper bound on $g(k, l)$ and use it for a better analysis of $T(k)$. In particular we are interested in the case when $l \geq k/2$.

Consider an out-tree T^0 on $k \geq 3$ vertices which has αk internal vertices and $(1 - \alpha)k$ leaves. We want to obtain an upper bound on the number of such non-isomorphic out-trees T^0. Let T^c be the subtree of T^0 obtained after deleting all its leaves and suppose that T^c has βk leaves. Assume that $\alpha \leq 1/2$ and notice that αk and βk are integers. Clearly $\beta < \alpha$.

Each out-tree T^0 with $(1 - \alpha)k$ leaves can be obtained by appending $(1 - \alpha)k$ leaves to T^c so that each of the vertices in Leaf(T^c) has at least one leaf appended to it. Imagine that we have $\beta k = |\text{Leaf}(T^c)|$ and $\alpha k - \beta k = |\text{Int}(T^c)|$ distinct boxes. Then what we are looking for is the number of ways to put $(1 - \alpha)k$ balls into the boxes so that each of the first βk boxes is nonempty. Again this is equivalent to putting $(1 - \alpha - \beta)k$ balls into αk distinct boxes. It is an easy exercise to see that this number equals $\binom{k-\beta k-1}{\alpha k-1}$.

Note that the above number does not give the exact value for the non-isomorphic out-trees on k vertices with $(1-\alpha)k$ leaves. This is because we treat an out-tree T^c as a labeled one, which may lead to us to distinguishing two assignments of balls even though the two corresponding out-trees T^0's are isomorphic to each other.

A minimal k-tree obtained from T^0 has $(1 - \alpha)k$ leaves and thus $(2 - \alpha)k$ vertices. With the upper bound $O^*(2.95^{\alpha k})$ on the number of T^c's by [100], by (7.2) we have the following:

$$\begin{aligned}
T(k) &= O^*\left(\sum_{\alpha\le 1/2}\sum_{\beta<\alpha} 2.95^{\alpha k}\binom{k-\beta k-1}{\alpha k-1}(6.14)^{(2-\alpha)k}\right) + O^*\left(\sum_{\alpha>1/2} 2.95^{\alpha k}(6.14)^{(2-\alpha)k}\right)\\
&\le O^*\left(\sum_{\alpha\le 1/2}\sum_{\beta<\alpha} 2.95^{\alpha k}\binom{k}{\alpha k}(6.14)^{(2-\alpha)k}\right) + O^*\left(2.95^k(6.14)^{3k/2}\right)\\
&\simeq O^*\left(\sum_{\alpha\le 1/2}\left(2.95^{\alpha}\frac{1}{\alpha^{\alpha}(1-\alpha)^{1-\alpha}}(6.14)^{(2-\alpha)}\right)^k\right) + O^*(44.9^k)
\end{aligned}$$

The term in the sum over $\alpha \le 1/2$ above is maximized when $\alpha = \frac{2.95}{2.95+6.14}$, which yields $T(k) = O^*(55.8^k)$. Thus, we conclude with the following theorem.

Theorem 5.6.4. *The problem* Directed k-Internal *is solvable in time* $O^*(55.8^k)$.

Part II

CSPs Parameterized Above Tight Lower Bounds

Chapter 6

Strictly Above/Below Expectation Method

In this chapter we introduce a probabilistic approach for kernelization called the *Strictly Above/Below Expectation* method, SABEM in short. Combining simple probabilistic arguments, this method turns out to be a powerful tool to prove fixed-parameter tractability of maximization (minimization, respectively) problems parameterized above (or below, respectively) tight lower (upper, respectively) bounds.

Let us briefly sketch how SABEM works for optimization problems parameterized above/below their tight bounds. Typically, we first apply some reduction rules to reduce the given problem Π to its special case Π'. Then we introduce a random variable X such that the answer to Π' is Yes if and only if X takes with positive probability a value greater or equal to the parameter κ. Choosing such a random variable is not always feasible, the Betweenness as an example. In that case, we introduce X with a weaker property.

Suppose we have a random variable X with the desired property. Under reasonable assumptions, we can show that with positive probability X takes a value large enough, say larger than $\sqrt{\mathbb{E}(X^2)}$. The remaining task is to exhibit a lower bound on $\mathbb{E}(X^2)$ in terms of the given instance size m. If $\mathbb{E}(X^2) \geq m$, it is implied that X takes a value larger than m with positive probability. Then we can conclude that if the given instance size is sufficiently large in comparison to k the instance is yes, and thus the size of no-instances is bounded by a function of k. In many cases, we obtain problem kernels of polynomial size.

Although the scheme of SABEM is simple, yet its application to concrete problems are not straightforward. For one thing, one needs to figure out a proper set of data reduction rules to reduce the given problem. Introducing an appropriate random variable X is nontrivial as well since X itself carries a lot of information on the structure of the problem as we shall see later. Another difficulty lies in the computation of $\mathbb{E}(X^2)$ or its lower bound.

In the subsequent sections, we introduce some probabilistic inequalities that are used for SABEM and present some kernelization results obtained via SABEM. The list of problems we tackle include (1) Linear Ordering, (2) Three special cases of Max-Lin2, (3) Betweenness, (4) Max-r-SAT and (5) (general) Boolean CSPs.

In this chapter all random variables are real. A random variable is *discrete* if its distribution function has a finite or countable number of positive increases. $\mathbb{P}(\cdot)$ and $\mathbb{E}[\cdot]$ denote probability and expectation, respectively.

6.1 Probabilistic Inequalities

A random variable X is *symmetric* if $-X$ has the same distribution function as X. If X is discrete, then X is symmetric if and only if $\mathbb{P}(X = a) = \mathbb{P}(X = -a)$ for each real a. Let X be a symmetric variable for which the first moment $\mathbb{E}(X)$ exists. Then $\mathbb{E}(X) = \mathbb{E}(-X) = -\mathbb{E}(X)$ and, thus, $\mathbb{E}(X) = 0$.

If X happens to be a symmetric random variable then the following simple inequality can be useful [106].

Lemma 6.1.1. *If X is a symmetric random variable and $\mathbb{E}(X^2)$ is finite, then*

$$\mathbb{P}(\, X \geq \sqrt{\mathbb{E}(X^2)}\,) > 0.$$

If X is not symmetric then the following lemma can be used instead.

Lemma 6.1.2. *Let X be a real random variable and suppose that its first, second and fourth moments satisfy $\mathbb{E}[X] = 0$, $\mathbb{E}[X^2] = \sigma^2 > 0$ and $\mathbb{E}[X^4] \leq c\sigma^4$, respectively, for some constant c. Then $\mathbb{P}(X > \frac{\sigma}{2\sqrt{c}}) > 0$.*

Showing that a random variable X meets the conditions of Lemma 6.1.2 involves computing the first, second and fourth moments of X. As the computation of higher moments get trickier, the following result from harmonic analysis is very useful.

Lemma 6.1.3 (Hypercontractive Inequality [25, 64])**.** *Let $f = f(x_1, \dots, x_n)$ be a polynomial of degree r in n variables $x_1, \dots, x_n$ each with domain $\{-1, 1\}$. Define a random variable X by choosing a vector $(\varepsilon_1, \dots, \varepsilon_n) \in \{-1, 1\}^n$ uniformly at random and setting $X = f(\varepsilon_1, \dots, \varepsilon_n)$. Then $\mathbb{E}[X^4] \leq 9^r \mathbb{E}[X^2]^2$.*

If $f = f(x_1, \dots, x_n)$ is a polynomial in n variables $x_1, \dots, x_n$ each with domain $\{-1, 1\}$, then it can be written as $f = \sum_{S \subseteq [n]} c_S \prod_{i \in S} x_i$, where $[n] = \{1, \dots, n\}$ and c_S is a real for each $S \subseteq [n]$. The following dual, in a sense, form of the Hypercontractive Inequality is proved in Section 6.3 (see an explanation after Lemma 6.3.3).

Theorem 6.1.4. *Let $f = f(x_1, \dots, x_n)$ be a polynomial in n variables $x_1, \dots, x_n$ each with domain $\{-1, 1\}$ such that $f = \sum_{S \subseteq [n]} c_S \prod_{i \in S} x_i$. Suppose that no variable x_i appears in more than $\rho \geq 2$ monomials of f. Define a random variable X by choosing a vector $(\varepsilon_1, \dots, \varepsilon_n) \in \{-1, 1\}^n$ uniformly at random and setting $X = f(\varepsilon_1, \dots, \varepsilon_n)$. Then $\mathbb{E}[X^4] \leq 2\rho^2 \mathbb{E}[X^2]^2$.*

6.2 Linear Ordering

Let $D = (V, A)$ be a digraph with no loops or parallel arcs in which every arc ij has a positive weight w_{ij}. The problem of finding an acyclic subdigraph of D of maximum weight is known as Linear Ordering. Let $n = |V|$ and consider a bijection $\alpha : V \to \{1, \dots, n\}$. Observe that the subdigraphs $(V, \{\, ij \in A : \alpha(i) < \alpha(j) \,\})$ and $(V, \{\, ij \in A : \alpha(i) > \alpha(j) \,\})$ are acyclic. Since the two subdigraphs contain all arcs of D, at least one of them has weight at least $W/2$, where $W = \sum_{ij \in A} w_{ij}$, the *weight* of D. Thus, $W/2$ is a lower bound on the maximum weight of an acyclic subdigraph of D. Consider a digraph D where for every arc ij of D there is also an arc ji of the same weight. Each maximum weight subdigraph of D has weight exactly $W/2$. Hence the lower bound $W/2$ is tight.

We consider the following parameterized version of Linear Ordering.

> Linear Ordering Above Tight Lower Bound (Linear Ordering$_{\text{TLB}}$ for short)
>
> *Instance:* A digraph $D = (V, A)$, each arc ij has an integral positive weight w_{ij}, and a positive integer k.
>
> *Parameter:* The integer k.
>
> *Question:* Is there an acyclic subdigraph of D of weight at least $W/2 + k$, where $W = \sum_{ij \in A} w_{ij}$?

In this section we will show that Linear Ordering$_{\text{TLB}}$ admits a kernel with $O(k^2)$ arcs; consequently the problem is fixed-parameter tractable. Note that if we allow weights to be positive reals, then we can show, similarly to the NP-completeness proof given in the next section, that Linear Ordering$_{\text{TLB}}$ is NP-complete already for $k = 1$.

Data Reduction Rules

Consider the following reduction rule:

Reduction Rule 2. *Assume D has a directed 2-cycle iji; if $w_{ij} = w_{ji}$ delete both arcs, if $w_{ij} > w_{ji}$ delete the arc ji and replace w_{ij} by $w_{ij} - w_{ji}$, and if $w_{ji} > w_{ij}$ delete the arc ij and replace w_{ji} by $w_{ji} - w_{ij}$.*

It is easy to check that the answer to Linear Ordering$_{\text{TLB}}$ for a digraph D is yes if and only if the answer to Linear Ordering$_{\text{TLB}}$ is yes for a digraph obtained from D using the reduction rule as long as possible. Note that applying Rule 2 as long as possible results in an oriented graph.

Kernelization

Let $D = (V, A)$ be an oriented graph, let $n = |V|$ and $W = \sum_{ij\in A} w_{ij}$. Consider a random bijection: $\alpha : V \to \{1, \dots, n\}$ and a random variable $X(\alpha) = \frac{1}{2}\sum_{ij\in A} \varepsilon_{ij}(\alpha)$, where $\varepsilon_{ij}(\alpha) = w_{ij}$ if $\alpha(i) < \alpha(j)$ and $\varepsilon_{ij}(\alpha) = -w_{ij}$, otherwise. It is easy to see that $X(\alpha) = \sum\{\, w_{ij} : ij \in A, \alpha(i) < \alpha(j)\,\} - W/2$. Thus, the answer to Linear Ordering$_{\text{TLB}}$ is yes if and only if there is a bijection $\alpha : V \to \{1, \dots, n\}$ such that $X(\alpha) \geq k$. Since $\mathbb{E}(\varepsilon_{ij}) = 0$, we have $\mathbb{E}(X) = 0$.

Let $W^{(2)} = \sum_{ij\in A} w_{ij}^2$. We will prove the following:

Lemma 6.2.1. $\mathbb{E}(X^2) \geq W^{(2)}/12$.

Proof. Let $N^+(i)$ and $N^-(i)$ denote the sets of out-neighbors and in-neighbors of a vertex i in D. By the definition of X,

$$4 \cdot \mathbb{E}(X^2) = \sum_{ij\in A} \mathbb{E}(\varepsilon_{ij}^2) + \sum_{ij,pq\in A} \mathbb{E}(\varepsilon_{ij}\varepsilon_{pq}), \tag{6.1}$$

where the second sum is taken over ordered pairs of distinct arcs. Clearly, $\sum_{ij\in A} \mathbb{E}(\varepsilon_{ij}^2) = W^{(2)}$. To compute $\sum_{ij,pq\in A} \mathbb{E}(\varepsilon_{ij}\varepsilon_{pq})$ we consider the following cases:

Case 1: $\{i, j\} \cap \{p, q\} = \emptyset$. Then ε_{ij} and ε_{pq} are independent and $\mathbb{E}(\varepsilon_{ij}\varepsilon_{pq}) = \mathbb{E}(\varepsilon_{ij})\mathbb{E}(\varepsilon_{pq}) = 0$.

Case 2a: $|\{i, j\} \cap \{p, q\}| = 1$ and $i = p$. Since the probability that $i < \min\{j, q\}$ or $i > \max\{j, q\}$ is 2/3, $\varepsilon_{ij}\varepsilon_{iq} = w_{ij}w_{iq}$ with probability $\frac{2}{3}$ and $\varepsilon_{ij}\varepsilon_{iq} = -w_{ij}w_{iq}$ with probability $\frac{1}{3}$. Thus, for every $i \in V$ we have $\sum_{ij,iq\in A} \mathbb{E}(\varepsilon_{ij}\varepsilon_{iq}) = \frac{1}{3}\sum\{\, w_{ij}w_{iq} : j \neq q \in N^+(i)\,\} = \frac{1}{3}(\sum_{j\in N^+(i)} w_{ij})^2 - \frac{1}{3}\sum_{j\in N^+(i)} w_{ij}^2$.

Case 2b: $|\{i, j\} \cap \{p, q\}| = 1$ and $j = q$. Similarly to Case 2a, we obtain $\sum_{ij,pj\in A} \mathbb{E}(\varepsilon_{ij}\varepsilon_{pj}) = \frac{1}{3}(\sum_{i\in N^-(j)} w_{ij})^2 - \frac{1}{3}\sum_{i\in N^-(j)} w_{ij}^2$.

Case 3a: $|\{i, j\} \cap \{p, q\}| = 1$ and $i = q$. Since $\varepsilon_{ij}\varepsilon_{pi} = w_{ij}w_{pi}$ with probability $\frac{1}{3}$ and $\varepsilon_{ij}\varepsilon_{pi} = -w_{ij}w_{pi}$ with probability $\frac{2}{3}$, we obtain $\sum_{ij,pi\in A} \mathbb{E}(\varepsilon_{ij}\varepsilon_{pi}) = -\frac{1}{3}\sum\{\, w_{ij}w_{pi} : j \in N^+(i),\ p \in N^-(i)\,\} = -\frac{1}{3}\sum_{j\in N^+(i)} w_{ij} \sum_{p\in N^-(i)} w_{pi}$.

Case 3b: $|\{i,j\}\cap\{p,q\}| = 1$ and $j = p$. Similarly to Case 3a, we obtain $\sum_{ij,jq\in A}\mathbb{E}(\varepsilon_{ij}\varepsilon_{jq}) = -\frac{1}{3}\sum_{i\in N^-(j)} w_{ij}\sum_{q\in N^+(j)} w_{jq}$.

Equation (6.1) and the subsequent computations imply that $4\cdot\mathbb{E}(X^2) = W^{(2)} + \frac{1}{3}(Q-R)$, where

$$Q = \sum_{i\in V}\left((\sum_{j\in N^+(i)} w_{ij})^2 - \sum_{j\in N^+(i)} w_{ij}^2 + (\sum_{j\in N^-(i)} w_{ji})^2 - \sum_{j\in N^-(i)} w_{ji}^2 \right),$$

and

$$R = 2\cdot\sum_{i\in V}(\sum_{j\in N^+(i)} w_{ij})(\sum_{j\in N^-(i)} w_{ji}).$$

By the inequality of arithmetic and geometric means, for each $i \in V$, we have

$$(\sum_{j\in N^+(i)} w_{ij})^2 + (\sum_{j\in N^-(i)} w_{ji})^2 - 2(\sum_{j\in N^+(i)} w_{ij})(\sum_{j\in N^-(i)} w_{ji}) \geq 0.$$

Therefore,

$$Q - R \geq -\sum_{i\in V}\sum_{j\in N^+(i)} w_{ij}^2 - \sum_{i\in V}\sum_{j\in N^-(i)} w_{ji}^2 = -2W^{(2)},$$

and $4\cdot\mathbb{E}(X^2) \geq W^{(2)} - 2W^{(2)}/3 = W^{(2)}/3$, implying $\mathbb{E}(X^2) \geq W^{(2)}/12$. □

Now we can prove the main result of this section.

Theorem 6.2.2. *The problem* Linear Ordering$_{\text{TLB}}$ *admits a kernel with $O(k^2)$ arcs.*

Proof. Let H be a digraph. We know that the answer to Linear Ordering$_{\text{TLB}}$ for H is yes if and only if the answer to Linear Ordering$_{\text{TLB}}$ is yes for a digraph D obtained from H using Reduction Rule 2 as long as possible. Observe that D is an oriented graph. Let $\mathcal{B}$ be the set of bijections from V to $\{1,\ldots,n\}$. Observe that $f : \mathcal{B} \to \mathcal{B}$ such that $f(\alpha(v)) = |V| + 1 - \alpha(v)$ for each $\alpha \in \mathcal{B}$ is a bijection. Note that $X(f(\alpha)) = -X(\alpha)$ for each $\alpha \in \mathcal{B}$. Therefore, $\mathbb{P}(X = a) = \mathbb{P}(X = -a)$ for each real a and, thus, X is symmetric. Thus, by Lemmas 6.1.1 and 6.2.1, we have $\mathbb{P}(\ X \geq \sqrt{W^{(2)}/12}\ ) > 0$. Hence, if $\sqrt{W^{(2)}/12} \geq k$, there is a bijection $\alpha : V \to \{1,\ldots,n\}$ such that $X(\alpha) \geq k$ and, thus, the answer to Linear Ordering$_{\text{TLB}}$ (for both D and H) is yes. Otherwise, $|A| \leq W^{(2)} < 12\cdot k^2$. □

We close this section by outlining how Theorem 6.2.2 can be used to actually find a solution to Linear Ordering$_{\text{TLB}}$ if one exists. Let (D,k) be an instance of Linear Ordering$_{\text{TLB}}$ where $D = (V,A)$ is a directed graph with integral positive arc-weights and $k \geq 1$ is an integer. Let W be the total weight of D. As discussed above, we may assume that D is an oriented graph. If $|A| < 12k^2$ then we can find a solution, if one exists, by trying all subsets $A' \subseteq A$, and testing whether (V,A') is acyclic and has weight at least $W/2 + k$; this search can be carried out in time $2^{O(k^2)}$. Next we assume $|A| \geq 12k^2$. We know by Theorem 6.2.2 that (D, k) is a YES-instance; it remains to find a solution.

For a vertex $i \in V$ let $d_D(i)$ denote its unweighted degree in D, i.e., the number of arcs (incoming or outgoing) that are incident with i. Consider the following reduction rule:

Reduction Rule 3. *If there is a vertex $i \in V$ with $|A| - 12k^2 \geq d_D(i)$, then delete i from D.*

Observe that by applying the rule we obtain again a YES-instance $(D - i, k)$ of LINEAR ORDERING$_{\text{TLB}}$ since $D - i$ has still at least $12k^2$ arcs. Moreover, if we know a solution D'_i of $(D-i, k)$, then we can efficiently obtain a solution D' of (D, k): if $\sum_{j \in N^+(i)} w_{ij} \geq \sum_{j \in N^-(i)} w_{ij}$ then we add i and all outgoing arcs $ij \in A$ to D'_i; otherwise, we add i and all incoming arcs $ji \in A$ to D'_i. After multiple applications of Rule 3 we are left with an instance (D_0, k) to which Rule 3 cannot be applied. Let $D_0 = (V_0, A_0)$. We pick a vertex $i \in V_0$ arbitrarily. If i has a neighbor j with $d_{D_0}(j) = 1$, then $|A_0| \leq 12k^2$, since $|A_0| - d_{D_0}(j) < 12k^2$. On the other hand, if $d_{D_0}(j) \geq 2$ for all neighbors j of i, then i has less than $2 \cdot 12k^2$ neighbors, since $D_0 - i$ has less than $12k^2$ arcs; thus $|A_0| < 3 \cdot 12k^2$. Therefore, as above, time $2^{O(k^2)}$ is sufficient to try all subsets $A'_0 \subseteq A_0$ to find a solution to the instance (D_0, k). Let n denote the input size of instance (D, k). Rule 3 can certainly be applied in polynomial time $n^{O(1)}$, and we apply it less than n times. Hence, we can find a solution to (D, k), if one exists, in time $n^{O(1)} + 2^{O(k^2)}$.

Recall that a kernelization reduces in polynomial time an instance (I, k) of a parameterized problem to a *decision-equivalent* instance (I', k'), its *problem kernel*, where $k' \leq k$ and the size of I' is bounded by a function of k. Solutions for (I, k) and solutions for (I', k') are possibly unrelated to each other. We call (I', k') a *faithful problem kernel* if from a solution for (I', k') we can construct a solution for (I, k) in time polynomial in $|I|$ and k. Clearly the above (D_0, k) is a faithful kernel.

6.3 Max-Lin2

Consider a system of m linear equations $e_1, \ldots, e_m$ in n variables $z_1, \ldots, z_n$ over GF(2), and suppose that each equation e_j has a positive integral weight w_j, $j = 1, \ldots, m$. The problem MAX-LIN2 asks for an assignment of values to the variables that maximizes the total weight of the satisfied equations. Let $W = w_1 + \cdots + w_m$.

To see that the total weight of the equations that can be satisfied is at least $W/2$, we merely observe that a uniform random assignment of values to the variables will satisfy any equation with probability .5. Thus there exists an assignment of values such that the total weight of satisfied equations is at least $W/2$. To see that the lower bound $W/2$ is tight, consider a system consisting of pairs of equations of the form $\sum_{i \in I} z_i = 1$ and $\sum_{i \in I} z_i = 0$ where both equations have the same weight.

The uniform random assignment procedure can be derandomized via conditional expectation as is described in [74]. We assign values to the variables $z_1, \ldots, z_n$ one by one and simplify the system after each assignment. When we wish to assign 0 or 1 to z_i, we consider all equations reduced to the form $z_i = b$, for a constant b. Let W' be the total weight of all such equations. We set $z_i := 0$, if the total weight of such equations is at least $W'/2$, and set $z_i := 1$, otherwise. If there are no equations of the form $z_i = b$, we set $z_i := 0$.

Henceforth, we consider the following parameterized version of Max-Lin2.

> Max-Lin2 Parameterized Above Tight Lower Bound (Max-Lin2$_{\text{TLB}}$ for short)
>
> *Instance:* A system S of m linear equations $e_1, \ldots, e_m$ in n variables $z_1, \ldots, z_n$ over GF(2), each equation e_i with a positive integral weight w_i, $i = 1, 2, \ldots, m$, and a positive integer k. Each equation e_j can be written as $\sum_{i \in I_j} z_i = b_j$, where $\emptyset \neq I_j \subseteq \{1, \ldots, n\}$.
>
> *Parameter:* The integer k.
>
> *Question:* Is there an assignment of values to the variables $z_1, \ldots, z_n$ such that the total weight of the satisfied equations is at least $W/2 + k$, where $W = \sum_{i=1}^{m} w_i$?

Let r_j be the number of variables in equation e_j, and let $r(S) = \max_{i=1}^{m} r_j$. We are not able to determine whether Max-Lin2$_{\text{TLB}}$ is fixed-parameter tractable or not, but we can prove that the following three special cases are fixed-parameter tractable:

1. there is a set U of variables such that each equation contains an odd number of variables from U
2. there is a constant r such that $r(S) \leq r$
3. there is a constant ρ such that any variable appears in at most ρ equations

Notice that in our formulation of Max-Lin2$_{\text{TLB}}$ it is required that each equation has a positive integral weight. In a relaxed setting in which an equation may have any positive rational number as its weight, the problem is NP-complete even for $k = 1$ and each $r_j = 2$. Indeed, let each linear equation be of the form $z_u + z_v = 1$. Then the problem is equivalent to MaxCut, the problem of finding a cut of total weight at least L in an undirected graph G, where $V(G)$ is the set of variables, $E(G)$ contains (z_u, z_v) if and only if there is a linear equation $z_u + z_v = 1$, and the weight of an edge (z_u, z_v) equals the weight of the corresponding linear equation. The problem MaxCut is a well-known NP-complete problem.

Let us transform an instance I of MaxCut into an instance I' of the "relaxed" Max-Lin2$_{\text{TLB}}$ by replacing the weight w_i by $w'_i := w_i/(L - W/2)$. We may assume that $L - W/2 > 0$ since otherwise the instance is immediately seen as a yes-instance. Observe that the new instance I' has an assignment of values with total weight at least $W'/2 + 1$ if and only if I has a cut with total weight at least L. We are done.

Data Reduction Rules

Let A be the matrix of the coefficients of the variables in S. It is well-known that the maximum number of linearly independent columns of A equals $\text{rank}A$, and such a collection of columns can be found in time polynomial in n and m, using, e.g., the Gaussian elimination on columns [21]. We have the following reduction rule and supporting lemma.

Reduction Rule 4. *Let A be the matrix of the coefficients of the variables in S, let $t = \text{rank}A$ and let columns $a^{i_1}, \dots, a^{i_t}$ of A be linearly independent. Then set all variables not in $\{z_{i_1}, \dots, z_{i_t}\}$ to 0 and simplify the equations of S.*

Lemma 6.3.1. *Let T be obtained from S by Rule 4. Then T is a* yes-*instance if and only if S is a* yes-*instance. Moreover, T can be obtained from S in time polynomial in n and m.*

Proof. If $t = n$, set $T := S$, so assume that $t < n$. The remark before the lemma immediately implies that T can be obtained from S in time polynomial in n and m. Let S' be a system of equations from S and let T' be the corresponding system of equations from T. It is sufficient to prove the following claim:

There is an assignment of values to $z_1, \dots, z_n$ satisfying all equations in S' and falsifying the rest of equations in S if and only if there is an assignment of values to $z_{i_1}, \dots, z_{i_t}$ satisfying all equations in T' and falsifying the rest of equations in T.

Let an assignment z^0 of values to $z = (z_1, \dots, z_n)$ satisfy all equations of S' and falsify the equations of S'', where $S'' = S \setminus S'$. This assignment satisfies all equations of R, the system obtained from S by replacing the right hand side b_j of each equation in S'' by $1 - b_j$. Note that R has the same matrix A of coefficients with columns $a^1, \dots, a^n$. Let a column $a^i \notin \{a^{i_1}, \dots, a^{i_t}\}$. Then, by definition of $a^{i_1}, \dots, a^{i_t}$, $a^i = \lambda_1 a^{i_1} + \cdots + \lambda_t a^{i_t}$ for some numbers $\lambda_j \in \{0, 1\}$. Knowing the numbers λ_j, we may eliminate a variable z_i from R by replacing a^i with the sum of all columns from $\{a^{i_1}, \dots, a^{i_t}\}$ for which $\lambda_j = 1$ and carrying out the obvious simplification of the system. Thus, we may eliminate from R all variables $z_i \notin \{z_{i_1}, \dots, z_{i_t}\}$ and get $y_{i_1} a^{i_1} + \cdots + y_{i_t} a^{i_t} = b'$, where b' is the right hand side of R and each $y_j \in \{0, 1\}$. Now replace, in the modified R, the right hand side b'_j of each equation corresponding to an equation in S'' by $1 - b'_j$ obtaining T. Clearly, $(y_{i_1}, \dots, y_{i_t})$ satisfies all equations of T' and falsifies all equations in $T'' = T \setminus T'$.

Suppose now that $(y_{i_1}, \dots, y_{i_t})$ satisfies all equations of T' and falsifies all equations in T''. Then $(y_1, \dots, y_n)$, where $y_j = 0$ if $j \notin \{i_1, \dots, i_t\}$, satisfies all equations of S' and falsifies all equations in S''. Thus, the claim has been proved. □

Consider the following reduction rule for Max-Lin2$_{\text{TLB}}$.

Reduction Rule 5. *If we have, for a subset I of $\{1, 2, \dots, n\}$, the equation $\sum_{i \in I} z_i = b'$ with weight w', and the equation $\sum_{i \in I} z_i = b''$ with weight w'', then we replace this pair by one of these equations with weight $w' + w''$ if $b' = b''$ and, otherwise, by the equation whose weight is bigger, modifying its new weight to be the difference of the two old ones. If the resulting weight is 0, we omit the equation from the system.*

If Rule 5 is not applicable to a system we call the system *reduced under Rule 5*. Note that the problem Max-Lin2$_{\text{TLB}}$ for S and the system obtained from S by applying Rule 5 as long as possible have the same answer.

Kernelization

Let $I_j \subseteq \{1, \dots, n\}$ be the set of indices of the variables participating in equation e_j, and let $b_j \in \{0, 1\}$ be the right hand side of e_j. Define a random variable $X = \sum_{j=1}^{m} X_j$, where $X_j = (-1)^{b_j} w_j \prod_{i \in I_j} \varepsilon_i$ and all the ε_i are independent uniform random variables on $\{-1, 1\}$ (X was first introduced in [74]). We set $z_i = 0$ if $\varepsilon_i = 1$ and $z_i = 1$, otherwise, for each i. In other words, $\varepsilon_i = (-1)^{z_i}$. Then z_i are independent uniform random variables on $\{0, 1\}$ and observe that $X_j = w_j$ if e_j is satisfied and $X_j = -w_j$, otherwise. Note that the relation $\varepsilon_i = (-1)^{z_i}$ is well-known for Fourier expansions of pseudo-boolean functions, i.e., functions $f : \{-1, +1\}^n \to \mathbb{R}$, see, e.g., [97, 44].

Lemma 6.3.2. *Let S be reduced under Rule 5. The weight of the satisfied equations is at least $W/2 + k$ if and only if $X \geq 2k$. We have $\mathbb{E}(X) = 0$ and $\mathbb{E}(X^2) = \sum_{j=1}^{m} w_j^2$.*

Proof. Observe that X is the difference between the weights of satisfied and falsified equations. Therefore, the weight of the satisfied equations equals $(X + W)/2$, and it is at least $W/2 + k$ if and only if $X \geq 2k$. Since ε_i are independent, $\mathbb{E}(\prod_{i \in I_j} \varepsilon_i) = \prod_{i \in I_j} \mathbb{E}(\varepsilon_i) = 0$. Thus, $\mathbb{E}(X_j) = 0$ and $\mathbb{E}(X) = 0$ by linearity of expectation. Moreover,

$$\mathbb{E}(X^2) = \sum_{j=1}^{m} \mathbb{E}(X_j^2) + \sum_{1 \leq j \neq q \leq m} \mathbb{E}(X_j X_q) = \sum_{j=1}^{m} w_j^2 > 0$$

as $\mathbb{E}(\prod_{i \in I_j} \varepsilon_i \cdot \prod_{i \in I_q} \varepsilon_i) = \mathbb{E}(\prod_{i \in I_j \Delta I_q} \varepsilon_i) = 0$ implies $\mathbb{E}(X_j X_q) = 0$, where $I_j \Delta I_q$ is the symmetric difference between I_j and I_q ($I_j \Delta I_q \neq \emptyset$ due to Rule 5). □

Lemma 6.3.3. *Let S be reduced under Rule 5 and suppose that no variable appears in more than $\rho \geq 2$ equations of S. Then $\mathbb{E}(X^4) \leq 2\rho^2(\mathbb{E}(X^2))^2$.*

Proof. Observe that

$$\mathbb{E}(X^4) = \sum_{(p,q,s,t)\in[m]^4} \mathbb{E}(X_pX_qX_sX_t), \tag{6.2}$$

where $[m] = \{1,\dots,m\}$. Note that if the product $X_pX_qX_sX_t$ contains a variable ε_i in only one or three of the factors, then $\mathbb{E}(X_pX_qX_sX_t) = A\cdot\mathbb{E}(\varepsilon_i) = 0$, where A is a polynomial in random variables ε_l, $l\in\{1,\dots,n\}\setminus\{i\}$. Thus, the only nonzero terms in (6.2) are those for which either (1) $p=q=s=t$, or (2) there are two distinct integers j,l such that each of them coincides with two elements in the sequence p,q,s,t, or (3) $|\{p,q,s,t\}|=4$, but each variable ε_i appears in an even number of the factors in $X_pX_qX_sX_t$. In Cases 1 and 2, we have $\mathbb{E}(X_pX_qX_sX_t) = w_p^4$ and $\mathbb{E}(X_pX_qX_sX_t) = w_j^2w_l^2$, respectively. In Case 3,

$$\mathbb{E}(X_pX_qX_sX_t) \le w_pw_qw_sw_t \le (w_p^2w_q^2 + w_s^2w_t^2)/2.$$

Let $1\le j<l\le m$. Observe that $\mathbb{E}(X_pX_qX_sX_t) = w_j^2w_l^2$ in Case 2 for $\binom{4}{2} = 6$ 4-tuples $(p,q,s,t)\in[m]^4$. In Case 3, we claim that $j,l\in\{p,q,s,t\}$ for at most $4\cdot(\rho-1)^2$ 4-tuples $(p,q,s,t)\in[m]^4$. To see this, first note that $w_p^2w_q^2$ and $w_s^2w_t^2$ appear in our upper bound on $\mathbb{E}(X_pX_qX_sX_t)$ (with coefficient 1/2). Therefore, there are only four possible ways for $w_j^2w_l^2$ to appear in our upper bound, namely the following: (i) $j=p, l=q$, (ii) $l=p, j=q$, (iii) $j=s, l=t$, and (iv) $l=s, j=t$. Now assume, without loss of generality, that $j=p$ and $l=q$. Since S is reduced under Rule 5, the product X_jX_l must have a variable ε_i of degree one. Thus, ε_i must be in X_s or X_t, but not in both (two choices). Assume that ε_i is in X_s. Observe that there are at most $\rho-1$ choices for s. Note that $X_jX_lX_s$ must contain contain a variable $\varepsilon_{i'}$ of odd degree. Thus, $\varepsilon_{i'}$ must be in X_t and, hence, there are at most $\rho-1$ choices for t.

Therefore, we have

$$\mathbb{E}(X^4) \le \sum_{j=1}^{m} w_j^4 + (6+4(\rho-1)^2)\sum_{1\le j<l\le m} w_j^2w_l^2 < 2\rho^2\left(\sum_{j=1}^{m} w_j^2\right)^2.$$

Thus, by Lemma 6.3.2, $\mathbb{E}(X^4)\le 2\rho^2(\mathbb{E}(X^2))^2$. □

Observe that Lemma 6.3.3 and the relation $\varepsilon_i = (-1)^{z_i}$, described before Lemma 6.3.2 between weighted systems of linear equations on GF(2) and n-variate polynomials with domain $\{-1,1\}^n$, imply immediately Theorem 6.1.4 (essentially Theorem 6.1.4 and Lemma 6.3.3 are equivalent via the relation).

Now we can prove the following:

Theorem 6.3.4. *Let S be reduced under Rule 5. The following three special cases of* Max-Lin2$_{\text{TLB}}$ *are fixed-parameter tractable: (1) there is a set U of variables such that* *each equation contains an odd number of variables from U, (2) there is a constant r such that $r(S) \leq r$, (3) there is a constant ρ, such that any variable appears in at most ρ equations. In each case, there exists a kernel with $O(k^2)$ equations and variables.*

Proof. **Case 1.** Due to the relation $\varepsilon_i = (-1)^{z_i}$ we may consider X as a random variable depending on random variables $z_1, \ldots, z_n$. Let $z^0 = (z_1^0, \ldots, z_n^0) \in \{0,1\}^n$ be an assignment of values to the variables $z_1, \ldots, z_n$, and let $-z^0 = (z_1', \ldots, z_n')$, where $z_i' = 1 - z_i^0$ if $z_i \in U$ and $z_i' = z_i^0$, otherwise, $i = 1, \ldots, n$. Observe that $f: z^0 \mapsto -z^0$ is a bijection on the set of assignments and $X(-z^0) = -X(z^0)$. Thus, X is a symmetric random variable. Therefore, by Lemmas 6.1.1 and 6.3.2, $\mathbb{P}(X \geq \sqrt{m}) \geq \mathbb{P}(X \geq \sqrt{\sum_{j=1}^m w_j^2}) > 0$. Hence, if $\sqrt{m} \geq 2k$, the answer to Max-Lin2$_{\text{TLB}}$ is YES. Otherwise, $m < 4k^2$ and after applying Rule 4, we obtain a kernel with $O(k^2)$ equations and variables.

Case 2. Since X is a polynomial of degree at most r, it follows by Lemma 6.1.3 that $\mathbb{E}(X^4) \leq 9^r \mathbb{E}(X^2)^2$. This inequality and Lemma 6.3.2 show that the conditions of Lemma 6.1.2 are satisfied and, thus,

$$\mathbb{P}\left(X > \frac{\sqrt{\sum_{j=1}^m w_j^2}}{2 \cdot 3^r}\right) > 0, \quad \text{implying} \quad \mathbb{P}\left(X > \frac{\sqrt{m}}{2 \cdot 3^r}\right) > 0.$$

Consequently, if $2k - 1 \leq \sqrt{m}/(2 \cdot 3^r)$, then there is an assignment of values to the variables $z_1, \ldots, z_n$ which satisfies equations of total weight at least $W/2 + k$. Otherwise, $2k - 1 > \sqrt{m}/(2 \cdot 3^r)$ and $m < 4(2k-1)^2 9^r$. After applying Rule 4, we obtain the required kernel.

Case 3. If $\rho = 1$, it is easy to find an assignment to the variables that satisfies all equations of S. Thus, we may assume that $\rho \geq 2$. To prove that there exists a kernel with $O(k^2)$ equations, we can proceed as in Case 2, but use Lemma 6.3.3 rather than Lemma 6.1.3. □

Case 1 of Theorem 6.3.4 is of interest since its condition can be checked in polynomial time due to the following:

Proposition 6.3.5. *We can check, in polynomial time, whether there exists a set U of variables such that each equation of S contains an odd number of variables from U.*

Proof. Observe that such a set U exists if and only if the unweighted system S' of linear equations over GF(2) obtained from S by replacing each b_j with 1 has a solution. Indeed, if U exists, set $z_j = 1$ for each $z_j \in U$ and $z_j = 0$ for each $z_j \notin U$. This assignment is a solution to S'. If a solution to S' exists, form U by including in it all variables z_j

which equal 1 in the solution. We can check whether S' has a solution using the Gaussian elimination or other polynomial-time algorithms, see, e.g., [37]. □

Remark 1. Note that even if S does not satisfy Case 2 of the theorem, T, the system obtained from S using Rule 4, may still satisfy Case 2. However, we have not formulated the theorem for S reduced under Rule 4 as the reduced system depends on the choice of a maximum linear independent collection of columns of A.

6.4 Betweenness

We study the one-dimensional ordinal embedding of partial orders that specify the maximum edge for some triangles. This problem has been studied under the name of BETWEENNESS, which takes a set V of variables and a set C of *betweenness constraints* of the form "v_i is between v_j and v_k" for distinct variables $v_i, v_j, v_k \in V$. Such a constraint will be written as $(v_i, \{v_j, v_k\})$. The objective is to find a bijection α from V to the set $\{1, \ldots, |V|\}$ that "satisfies" the maximum number of constraints from C, where a constraint $(v_i, \{v_j, v_k\})$ is *satisfied* by α if either $\alpha(v_j) < \alpha(v_i) < \alpha(v_k)$ or $\alpha(v_k) < \alpha(v_i) < \alpha(v_j)$ holds. We also refer to α as a *linear arrangement* of V.

Notice that a uniformly random permutation of the variables in V satisfies one-third of the constraints in expectation and thus $|C|/3$ is a lower bound on any optimal solution. On the other hand, for a set C of constraints containing all three possible constraints on each 3-set of variables, no more than $|C|/3$ of the constraints in C can be satisfied in any linear arrangement. Hence the lower bound of one-third on the fraction of satisfiable constraints is tight, in the sense that it is attained by an infinite family of instances.

So the right question to ask is whether there exists a linear arrangement that satisfies at least $|C|/3 + \kappa$ of the constraints as given below.

> BETWEENNESS ABOVE TIGHT LOWER BOUND (BETWEENNESS$_{\text{TLB}}$)
>
> *Instance:* a set C of betweenness constraints over variables V and an integer $\kappa \geq 0$.
>
> *Parameter:* The integer κ.
>
> *Question:* Is there a bijection $\alpha : V \to \{1, \ldots, |V|\}$ that satisfies at least $|C|/3 + \kappa$ constraints from C, that is, for at least $|C|/3 + \kappa$ constraints $(v_i, \{v_j, v_k\}) \in C$ we have either $\alpha(v_j) < \alpha(v_i) < \alpha(v_k)$ or $\alpha(v_k) < \alpha(v_i) < \alpha(v_j)$?

Data Reduction Rule

For a constraint C of $\mathcal{C}$ let $vars(C)$ denote the set of variables in C. We call a triple A, B, C of distinct betweenness constraints *complete* if $vars(A) = vars(B) = vars(C)$.

Consider the following *reduction rule*: if $\mathcal{C}$ contains a complete triple of constraints, delete these constraints from $\mathcal{C}$ and delete from V any variable that appears only in the triple. Since for every linear arrangement exactly one constraint in each complete triple is satisfied we have the following:

Lemma 6.4.1. *Let* $(V, \mathcal{C})$ *be an instance of* BETWEENNESS$_{\text{TLB}}$ *and let* $(V', \mathcal{C}')$ *be obtained from* $(V, \mathcal{C})$ *by applying the reduction rule as long as possible. Then* $(V, \mathcal{C})$ *is a* YES-*instance of* BETWEENNESS$_{\text{TLB}}$ *if and only if so is* $(V', \mathcal{C}')$.

An instance $(V, \mathcal{C})$ of BETWEENNESS$_{\text{TLB}}$ is *irreducible* if it does not contain a complete triple. Observe that using Lemma 6.4.1 we can transform any instance into an irreducible one and it will take no more than $O(m^3)$ time.

Kernelization

Consider an instance $(V, \mathcal{C})$, for a set V of variables and a set $\mathcal{C} = \{C_1, \ldots, C_m\}$ of betweenness constraints, and a random function $\phi : V \to \{0, 1, 2, 3\}$. (The reason we consider a random function $\phi : V \to \{0, 1, 2, 3\}$ rather than a random function $\phi : V \to \{0, 1\}$ is given in the end of this section.) Let $\ell_i(\phi)$ be the number of variables in V mapped by ϕ to i for $i = 0, 1, 2, 3$. Now obtain a bijection $\alpha : V \to \{1, \ldots, |V|\}$ by randomly assigning values $1, \ldots, \ell_0(\phi)$ to all $\alpha(v)$ for which $\phi(v) = 0$, and values $\sum_{i=0}^{j-1} \ell_i(\phi) + 1, \ldots, \sum_{i=0}^{j} \ell_i(\phi)$ to all $\alpha(v)$ for which $\phi(v) = j$ for every $j = 1, 2, 3$. We call such a linear arrangement α a *ϕ-compatible* bijection. It is easy to see that α obtained in this two stage process is, in fact, a random linear arrangement, but this fact is not going to be used here.

Now assume that a function $\phi : V \to \{0, 1, 2, 3\}$ is fixed and consider a constraint $C_p = (v_i, \{v_j, v_k\}) \in \mathcal{C}$. Let α be a random ϕ-compatible bijection and $\nu_p(\alpha) = 1$ if C_p is satisfied and 0, otherwise. Let the *weights* $w(C_p, \phi) = \mathbb{E}(\nu_p(\alpha)) - 1/3$ and $w(\mathcal{C}, \phi) = \sum_{p=1}^{m} w(C_p, \phi)$.

Lemma 6.4.2. *If* $w(\mathcal{C}, \phi) \geq \kappa$ *then* $(V, \mathcal{C})$ *is a* YES-*instance of* BETWEENNESS$_{\text{TLB}}$.

Proof. By linearity of expectation, $w(\mathcal{C}, \phi) \geq \kappa$ implies $\mathbb{E}(\sum_{p=1}^{m} \nu_p(\alpha)) \geq m/3 + \kappa$. Thus, if $w(\mathcal{C}, \phi) \geq \kappa$ then there is a ϕ-compatible bijection α that satisfies at least $m/3 + \kappa$ constraints. □

Let $X = w(\mathcal{C}, \phi)$ and $X_p = w(C_p, \phi)$, $p = 1, \ldots, m$. Observe that if ϕ is a random function from V to $\{0, 1, 2, 3\}$ then $X, X_1, \ldots, X_m$ are random variables. Recall that $X = \sum_{p=1}^{m} X_p$.

Lemma 6.4.3. *We have* $\mathbb{E}[X] = 0$.

$\vert\{\phi(v_i), \phi(v_j), \phi(v_k)\}\vert$	Relation	Value of X_p	Prob.
1	$\phi(v_i) = \phi(v_j) = \phi(v_k)$	0	1/16
2	$\phi(v_i) \neq \phi(v_j) = \phi(v_k)$	$-1/3$	3/16
2	$\phi(v_i) \in \{\phi(v_j), \phi(v_k)\}$	1/6	6/16
3	$\phi(v_i)$ is between $\phi(v_j)$ and $\phi(v_k)$	2/3	2/16
3	$\phi(v_i)$ is not between $\phi(v_j)$ and $\phi(v_k)$	$-1/3$	4/16

Table 6.1: Distribution of X_p.

Proof. Let $C_p = (v_i, \{v_j, v_k\}) \in C$. Let us first find the distribution of X_p. It is easy to check that the probability that $\phi(v_i) = \phi(v_j) = \phi(v_k)$ equals 1/16 and $X_p = 0$ in such a case. The probability that $\phi(v_i) \neq \phi(v_j) = \phi(v_k)$ equals 3/16 and $X_p = -1/3$ in such a case. The probability that $\phi(v_i)$ equals one of the non-equal $\phi(v_j), \phi(v_k)$ is equal to 6/16 and $X_p = 1/6$ in such a case. Now suppose that $\phi(v_i), \phi(v_j)$ and $\phi(v_k)$ are all distinct. The probability that $\phi(v_i)$ is between $\phi(v_j)$ and $\phi(v_k)$ is 2/16 and $X_p = 2/3$ in such a case. Finally, the probability that $\phi(v_i)$ is not between $\phi(v_j)$ and $\phi(v_k)$ is 4/16 and $X_p = -1/3$ in such a case. Now we can give the distribution of X_p in the following table.

Using this distribution, it is easy to see that $\mathbb{E}[X_p] = 0$ and, thus, $\mathbb{E}[X] = \sum_{p=1}^{m} \mathbb{E}[X_p] = 0$. □

Lemma 6.4.4. *The random variable X can be expressed as a polynomial of degree 6 in independent uniformly distributed random variables with values -1 and 1.*

Proof. Consider $C_p = (v_i, \{v_j, v_k\}) \in C$. Let $\varepsilon_1^i = -1$ if $\phi(v_i) = 0$ or 1 and $\varepsilon_1^i = 1$, otherwise. Let $\varepsilon_2^i = -1$ if $\phi(v_i) = 0$ or 2 and $\varepsilon_2^i = 1$, otherwise. Similarly, we can define $\varepsilon_1^j, \varepsilon_2^j, \varepsilon_1^k, \varepsilon_2^k$. Now $\varepsilon_1^i\varepsilon_2^i$ can be seen as a binary representation of a number from the set $\{0, 1, 2, 3\}$ and $\varepsilon_1^i\varepsilon_2^i\varepsilon_1^j\varepsilon_2^j\varepsilon_1^k\varepsilon_2^k$ can be viewed as a binary representation of a number from the set $\{0, 1, \ldots, 63\}$, where -1 plays the role of 0.

We can write X_p as the following polynomial:

$$\frac{1}{64}\sum_{q=0}^{63}(-1)^{s_q} w_q \cdot (\varepsilon_1^i + c_1^{iq})(\varepsilon_2^i + c_2^{iq})(\varepsilon_1^j + c_1^{jq})(\varepsilon_2^j + c_2^{jq})(\varepsilon_1^k + c_1^{kq})(\varepsilon_2^k + c_2^{kq}),$$

where $c_1^{iq}c_2^{iq}c_1^{jq}c_2^{jq}c_1^{kq}c_2^{kq}$ is the binary representation of q, s_q is the number of digits equal -1 in this representation, and w_q equals the value of X_p for the case when the binary representations of $\phi(v_i), \phi(v_j)$ and $\phi(v_k)$ are $c_1^{iq}c_2^{iq}$, $c_1^{jq}c_2^{jq}$ and $c_1^{kq}c_2^{kq}$, respectively. The actual values for X_p for each case are given in the proof Lemma 6.4.3. The above polynomial is of degree 6. It remains to recall that $X = \sum_{p=1}^{m} X_p$. □

Lemma 6.4.5. *For an irreducible instance* (V, C) *of* Betweenness$_{\text{TLB}}$ *we have* $\mathbb{E}[X^2] \geq \frac{11}{768}m$.

Proof. First, observe that $\mathbb{E}[X^2] = \sum_{l=1}^{m} \mathbb{E}[X_l^2] + \sum_{1\leq l\neq l'\leq m} \mathbb{E}[X_l X_{l'}]$. We will compute $\mathbb{E}[X_l^2]$ and $\mathbb{E}[X_l X_{l'}]$ separately.

Using the distribution of X_l given in Table 1, it is easy to see that $\mathbb{E}[X_l^2] = 11/96 = 88/768$. It remains to show that

$$\sum_{1\leq l\neq l'\leq m} \mathbb{E}[X_l X_{l'}] \geq -\frac{77}{768}m. \tag{6.3}$$

Indeed, (7.2) and $\mathbb{E}[X_l^2] = 88/768$ imply that

$$\mathbb{E}[X^2] = \sum_{l=1}^{m} \mathbb{E}[X_l^2] + \sum_{1\leq l\neq l'\leq m} \mathbb{E}[X_l X_{l'}] \geq \frac{88}{768}m - \frac{77}{768}m = \frac{11}{768}m,$$

In the remainder of this proof we show that (7.2) holds. Let $C_l, C_{l'}$ be a pair of distinct constraints of C. To evaluate $\mathbb{E}[X_l X_{l'}]$, we consider several cases. A simple case is when the sets $vars(C_l)$ and $vars(C_{l'})$ are disjoint: then X_l and $X_{l'}$ are independent random variables and, thus, $\mathbb{E}[X_l X_{l'}] = \mathbb{E}[X_l]\mathbb{E}[X_{l'}] = 0$. Let $U = \{(l, l') \mid C_l, C_{l'} \in C, l \neq l'\}$ be the set of all ordered index pairs corresponding to distinct constraints in C. We will classify subcases of this case by considering some subsets of U. Let

$$\begin{aligned}
S_1(u) &= \{(l, l') \in U \ : \ C_l = (u, \{a, b\}), C_{l'} = (u, \{c, d\}),\ a, b, c, d \in V\},\\
S_2(u) &= \{(l, l') \in U \ : \ C_l = (a, \{u, b\}), C_{l'} = (c, \{u, d\}),\ a, b, c, d \in V\},\\
S_3(u) &= \{(l, l'), (l', l) \in U \ : \ C_l = (u, \{a, b\}), C_{l'} = (c, \{u, d\}),\ a, b, c, d \in V\},\\
S_4(u, v) &= \{(l, l') \in U \ : \ C_l = (u, \{v, a\}), C_{l'} = (u, \{v, b\}),\ a, b \in V\}\\
&\cup \ \{(l, l') \in U \ : \ C_l = (v, \{u, a\}), C_{l'} = (v, \{u, b\}),\ a, b \in V\},\\
S_5(u, v) &= \{(l, l') \in U \ : \ C_l = (a, \{u, v\}), C_{l'} = (b, \{u, v\}),\ a, b \in V\},\\
S_6(u, v) &= \{(l, l'), (l', l) \in U \ : \ C_l = (u, \{v, a\}), C_{l'} = (b, \{u, v\}),\ a, b \in V\}\\
&\cup \ \{(l, l'), (l', l) \in U \ : \ C_l = (v, \{u, a\}), C_{l'} = (b, \{u, v\}),\ a, b \in V\},\\
S_7(u, v) &= \{(l, l'), (l', l) \in U \ : \ C_l = (u, \{v, a\}), C_{l'} = (v, \{u, b\}),\ a, b \in V\}\\
S_8(u, v, w) &= \{(l, l') \in U \ : \ vars(C_l) = vars(C_{l'}) = \{u, v, w\}\}.
\end{aligned}$$

Let $u, v \in V$ be a pair of distinct variables. Observe that $S_4(u, v) = (S_1(u) \cap S_2(v)) \cup (S_1(v) \cap S_2(u))$, $S_5(u, v) = S_2(u) \cap S_2(v)$, $S_6(u, v) = (S_3(u) \cap S_2(v)) \cup (S_3(v) \cap S_2(u))$ and

Set	Union/intersection Form	$\lvert\text{Set}\rvert$	$768\mathbb{E}[X_lX_{l'}]$	$768w'$
$S_1(u)$	–	$b(u)(b(u)-1)$	$12 = w_1$	12
$S_2(u)$	–	$e(u)(e(u)-1)$	$3 = w_2$	3
$S_3(u)$	–	$b(u)e(u)+e(u)b(u)$	$-6 = w_3$	-6
$S_4(u,v)$	$(S_1(u)\cap S_2(v))\cup(S_1(v)\cap S_2(u))$	$c^u_v(c^u_v-1)+c^v_u(c^v_u-1)$	$24 = w_4$	9
$S_5(u,v)$	$S_2(u)\cap S_2(v)$	$c_{uv}(c_{uv}-1)$	$36 = w_5$	30
$S_6(u,v)$	$(S_3(u)\cap S_2(v))\cup(S_3(v)\cap S_2(u))$	$2(c^u_v+c^v_u)\cdot c_{uv}$	$-18 = w_6$	-15
$S_7(u,v)$	$S_3(u)\cap S_3(v)$	$2c^u_vc^v_u$	$-6 = w_7$	6
$S_8(u,v,w)$	see (7.3)	≤ 2	$-44 = w_8$	-11

Table 6.2: Data for sets $S_i(u)$, $i = 1,2,\dots,8$.

$S_7(u,v) = S_3(u)\cap S_3(v)$. Let $u,v,w \in V$ be a triple of distinct variables. Observe that

$$S_8(u,v,w) = (S_3(u)\cap S_3(v)\cap S_2(w))\cup(S_3(v)\cap S_3(w)\cap S_2(u))\cup(S_3(w)\cap S_3(u)\cap S_2(v)). \tag{6.4}$$

For a variable $u \in V$, let $b(u) = |\{l : C_l = (u,\{a,b\}), a,b \in V\}|$ and $e(u) = |\{l : C_l = (a,\{u,b\}), a,b \in V\}|$. Observe that $|S_1(u)| = b(u)(b(u)-1)$, $|S_2(u)| = e(u)(e(u)-1)$ and $|S_3(u)| = 2b(u)e(u)$.

For a pair $u,v \in V$, let $c^u_v = |\{l : C_l = (u,\{v,a\}), a \in V\}|$ and $c_{uv} = |\{l : C_l = (a,\{u,v\}), a \in V\}|$. Observe that $|S_4(u,v)| = c^u_v(c^u_v-1)+c^v_u(c^v_u-1)$, $|S_5(u,v)| = c_{uv}(c_{uv}-1)$, $|S_6(u,v)| = 2(c^u_v+c^v_u)\cdot c_{uv}$ and $|S_7(u,v)| = 2c^u_vc^v_u$. Let $u,v,w \in V$ be a triple of distinct variables. Since C is irreducible, the number of ordered pairs $(C_l, C_{l'})$ for which $vars(C_l) = vars(C_{l'}) = \{u,v,w\}$ is at most 2, i.e., $|S_8(u,v,w)| \leq 2$.

We list the sets $S_i(\cdot)$, their union/intersection forms (for $i = 4,5,6,7$) and their sizes in Table 2. If (l,l') belongs to some S_i but to no S_j for $j > i$, then Table 2 also contains the value $768\cdot\mathbb{E}[X_lX_{l'}]$, in the row corresponding to S_i. These values cannot be easily calculated analytically as there are many cases to consider and we have calculated them using computer. We will briefly describe how our program computes $\mathbb{E}[X_lX_{l'}]$ using as an example the case $(l,l') \in S_1(u)$, i.e., $C_l = (u,\{a,b\}), C_{l'} = (u,\{c,d\})$. For each $(q_1,q_2,q_3,q_4,q_5) \in \{0,1,2,3\}^5$ the probability of $(u,a,b,c,d) = (q_1,q_2,q_3,q_4,q_5)$ is 4^{-5} and the corresponding value of $X_lX_{l'}$ can be found in Table 2.

We are now ready to compute a lower bound on the term $\sum_{1\leq l\neq l'\leq m}\mathbb{E}[X_lX_{l'}]$. Define the values w'_i for $i = 1,2,\dots,8$ as it is done in Table 2. We will now show that the following holds (note that the sets we sum over have to contain distinct elements).

$$\begin{aligned}\textstyle\sum_{1\le l\neq l'\le m}\mathbb{E}[X_lX_{l'}] &= \textstyle\sum_{u\in V}\sum_{i=1}^{3}|S_i(u)|w'_i + \sum_{\{u,v\}\subseteq V}\sum_{i=4}^{7}|S_i(u,v)|w'_i\\ &\quad \textstyle+ \sum_{\{u,v,w\}\subseteq V}|S_8(u,v)|w'_8\end{aligned}$$

In order to show the above we consider the possible cases for $(l,l')\in U$.

Case 1: $|vars(C_l)\cap vars(C_{l'})| = 0$. In this case $\mathbb{E}[X_lX_{l'}] = 0$ and the corresponding (l,l') does not belong to any S_i and therefore contributes zero to the right-hand side above.

Case 2: $|vars(C_l)\cap vars(C_{l'})| = 1$. Each pair $(l,l')\in S_1(u)$ contributes $\frac{12}{768}$ to both sides of the above equation, as in this case (l,l') does not belong to any S_j with $j>1$. Analogously if $(l,l')\in S_2(u)$ then it contributes $\frac{3}{768}$ to both sides of the above equation. Furthermore if $(l,l')\in S_3(u)$ then it contributes $-\frac{6}{768}$.

Case 3: $|vars(C_l)\cap vars(C_{l'})| = 2$. Consider a pair $(l,l')\in S_4(u,v)$ and assume, without loss of generality, that $(l,l')\in S_1(u)\cap S_2(v)$. Note that (l,l') contributes $\frac{24}{768}$ to the left-hand side of the equation and it contributes $w'_1+w'_2+w'_4=\frac{24}{768}$ to the right-hand side (as $(l,l')\in S_1(u)\cap S_2(v)\cap S_4(u,v)$). Analogously if $(l,l')\in S_5(u,v)$ we get a contribution of $w_5=\frac{36}{768}=w'_2+w'_2+w'_5$ to both sides of the equation. If $(l,l')\in S_6(u,v)$ we get a contribution of $w_6=-\frac{18}{768}=w'_3+w'_2+w'_6$ to both sides of the equation. If $(l,l')\in S_7(u,v)$ we get a contribution of $w_7=-\frac{6}{768}=w'_3+w'_3+w'_7$ to both sides of the equation.

Case 4: $|vars(C_l)\cap vars(C_{l'})| = 3$. Assume, without loss of generality, that $(l,l')\in S_3(u)\cap S_3(v)\cap S_2(w)$ and note that $(l,l')\in S_7(u,v)\cap S_6(u,w)\cap S_6(v,w)$. Therefore we get a contribution of $w_8=-\frac{44}{768}=w'_3+w'_3+w'_2+w'_7+w'_6+w'_6+w'_8$ to both sides of the equation.

Therefore the above equation holds, which implies the following:

$$\begin{aligned}\textstyle\sum_{1\le l\neq l'\le m}\mathbb{E}[X_lX_{l'}] &= \textstyle\sum_{u\in V}\Big(|S_1(u)|w'_1+|S_2(u)|w'_2+|S_3(u)|w'_3\Big)\\ &\quad\textstyle+\sum_{\{u,v\}\subseteq V}\sum_{i=4}^{7}|S_i(u,v)|w'_i+\sum_{\{u,v,w\}\subseteq V}|S_8(u,v,w)|w'_8\\ &= \textstyle\frac{1}{2\cdot768}\sum_{u\in V}\Big(6(2b(u)-e(u))^2-24b(u)-6e(u)\Big)\\ &\quad\textstyle+\frac{1}{2\cdot768}\sum_{\{u,v\}\subseteq V}\left(15(c_v^u+c_u^v-2c_{uv})^2+12\left(\frac{c_v^u-c_u^v}{2}\right)^2-18(c_v^u+c_u^v)-60c_{uv}\right)\\ &\quad\textstyle+\sum_{\{u,v,w\}\subseteq V}|S_8(u,v,w)|w'_8\end{aligned}$$

To complete the proof of the lemma it remains to translate this sum into a function on the number of constraints. In that respect, notice that $\sum_{u\in V}b(u)=m$ and $\sum_{u\in V}e(u)=2m$. Further, each clause $(u,\{v,w\})$ contributes exactly one unit to each of c_v^u and c_w^u, as well as exactly one unit to c_{vw}. Hence $\sum_{\{u,v\}\subset V}(c_v^u+c_u^v)=2m$ and $\sum_{\{u,v\}\subset V}c_{uv}=m$. Since C is irreducible, the number of ordered pairs $(C_l,C_{l'})$ for which $vars(C_l)=vars(C_{l'})$ is at most

$m/2$ and, thus,

$$\sum_{\{u,v,w\}\subseteq V} |S_8(u,v,w)|w'_8 \leq m \cdot w'_8.$$

Together these bounds imply that

$$\sum_{1\leq l\neq l'\leq m} \mathbb{E}[X_l X_{l'}] \geq -\frac{36}{2\cdot 768}m - \frac{96}{2\cdot 768}m - \frac{11}{768}m = -\frac{77}{768}m$$

and (7.2) holds. □

We are now ready to prove the main result.

Theorem 6.4.6. Betweenness$_{\text{TLB}}$ *has a kernel of size* $O(\kappa^2)$.

Proof. Let (V,C) be an instance of Betweenness$_{\text{TLB}}$. By Lemma 6.4.1, in time $O(m^3)$ we can obtain an irreducible instance (V',C') such that (V,C) is a Yes-instance if and only if (V',C') is a Yes-instance. Let $m' = |C'|$ and let X be the random variable defined above. Then X is expressible as a polynomial of degree 6 by Lemma 6.4.4; hence it follows from Lemma 6.1.3 that $\mathbb{E}[X^4] \leq 9^6\mathbb{E}[X^2]^2$. Consequently, X satisfies the conditions of Lemma 6.1.2, from which we conclude in combination with Lemma 6.4.5 that $\mathbb{P}\left(X > \frac{1}{2\cdot 9^3}\sqrt{\frac{11}{768}m'}\right) > 0$. By Lemma 6.4.2 if $\frac{1}{2\cdot 9^3}\sqrt{\frac{11}{768}m'} \geq \kappa$ then (V',C') is a Yes-instance for Betweenness$_{\text{TLB}}$. Otherwise, we have $m' = O(\kappa^2)$. This concludes the proof of the theorem. □

We complete this section by answering the following natural question: why have we considered functions $\phi : V \to \{0,1,2,3\}$ rather than functions $\phi : V \to \{0,1\}$? The latter would involve less computations and give a smaller degree of the polynomial representing X. The reason is that our proof of Lemma 6.4.5 would not work for functions $\phi : V \to \{0,1\}$ (we would only be able to prove that $\mathbb{E}[X^2] \geq \sum_{\{u,v\}\subset V}[c_u^v + c_v^u - 2c_{uv}]^2$, which is not enough).

6.5 MAX-r-SAT

We assume an infinite supply of propositional *variables*. A *literal* is a variable x or its negation $\bar{x}$. A *clause* is a finite set of literals not containing a complementary pair x and $\bar{x}$. A clause is of *size* r if it contains exactly r literals. For simplicity of presentation, we will denote a clause by a sequence of its literals. For example, the clause $\{\bar{x}, y\}$ will be denoted $\bar{x}y$ or equivalently $y\bar{x}$. A *CNF formula* F is a finite multiset of clauses (a clause may appear in the multiset several times). A variable x *occurs* in a clause if the clause contains x or $\bar{x}$, and x occurs in a CNF formula F if it occurs in some clause of F. Let var(C) and var(F) denote the sets of variables occurring in C and F, respectively. A CNF formula is an r-CNF formula if $|C| = r$ for all $C \in F$. Thus we require that each clause of a r-CNF formula contains *exactly* r different literals (some authors use for that the term *exact* r-CNF). A *truth assignment* is a mapping $\tau : V \to \{-1, 1\}$ defined on some set V of variables. In order to obtain a 'normalized' algebraic representation, we use $\{-1, 1\}$ instead of the usual $\{0, 1\}$ binary symbols. We write 2^V to denote the set of all truth assignments on V. A truth assignment τ *satisfies* a clause C if there is some variable $x \in C$ with $\tau(x) = 1$ or a negated variable $\overline{x} \in C$ with $\tau(x) = -1$. We write sat(τ, F) for the number of clauses of F that are satisfied by τ, and we write

$$\mathrm{sat}(F) = \max_{\tau \in 2^{\mathrm{var}(F)}} \mathrm{sat}(\tau, F).$$

In the classic optimization problem Max-r-Sat, the task is to find a truth assignment to the variables of a given r-CNF formula so as to satisfy as many clauses as possiblem. We shall consider the following parameterized version of Max-r-Sat.

> Max-r-Sat above Tight Lower Bound (or Max-r-Sat$_{\text{TLB}}$ for short)
>
> *Instance:* A pair (F, k) where F is a multiset of m clauses of size r and k is a nonnegative integer.
>
> *Parameter:* The integer k.
>
> *Question:* Is $\mathrm{sat}(F) \geq ((2^r - 1)m + k)/2^r$?

In this section we first describe a polynomial-time data reduction that reduces an instance of Max-r-Sat$_{\text{TLB}}$ into an equivalent algebraically represented problem. The equivalent algebraically represented problems is 'normalized' in a sense, which enables us to obtain a bound on the size of a given instance. Some results from probability theory and hamonic analysis in boolean functions play a central role in proving such a bound. As a result, we prove that Max-r-Sat$_{\text{TLB}}$ is fixed-parameter tractability and in particular we present a quadratic kernel using the notion of bikernelization introduced in the previous section.

An Algebraic Representation of Max-r-Sat$_{\text{TLB}}$

Let F be an r-CNF formula with clauses $C_1, \ldots, C_m$ in the variables $x_1, x_2, \ldots, x_n$. For F, consider

$$X = \sum_{C \in F} [1 - \prod_{x_i \in \mathrm{var}(C)} (1 + \varepsilon_i x_i)],$$

where $\varepsilon_i \in \{-1, 1\}$ and $\varepsilon_i = -1$ if and only if x_i is in C.

Lemma 6.5.1. *For a truth assignment τ, we have $X = 2^r(\mathrm{sat}(\tau, F) - (1 - 2^{-r})m)$.*

Proof. Observe that $\prod_{x_i \in \mathrm{var}(C)}(1 + \varepsilon_i x_i)$ equals 2^r if C is falsified and 0, otherwise. Thus, $X = m - 2^r(m - \mathrm{sat}(\tau, F))$ implying the claimed formula. □

After algebraic simplification $X = X(x_1, x_2, \ldots, x_n)$ can be written as $X = \sum_{I \in \mathcal{S}} X_I$, where $X_I = c_I \prod_{i \in I} x_i$, each c_I is a nonzero integer and $\mathcal{S}$ is a family of nonempty subsets of $\{1, \ldots, n\}$ each with at most r elements.

The question we address is that of deciding whether or not there are values $x_i \in \{-1, 1\}$ so that $X = X(x_1, x_2, \ldots, x_n) \geq k$. The idea is to use a probabilistic argument and show that if the above polynomial has many nonzero coefficients, that is, if $|\mathcal{S}|$ is large, this is necessarily the case, whereas if it is small, the question can be solved by checking all possibilities of the relevant variables.

Kernelization

Theorem 6.5.2. *The problem* Max-r-Sat$_{\text{TLB}}$ *is fixed-parameter tractable and can be solved in time $O(m) + 2^{O(k^2)}$. Moreover, there exist (i) a polynomial-size bikernel from* Max-r-Sat$_{\text{TLB}}$ *to* Max-Lin2$_{\text{TLB}}$*, and (ii) a polynomial-size kernel of* Max-r-Sat$_{\text{TLB}}$*. In fact, there are such a bikernel and a kernel of size $O(k^2)$.*

Proof. By Lemma 6.5.1 our problem is equivalent to that of deciding whether or not there is a truth assignment to the variables $x_1, x_2, \ldots, x_n$, so that

$$X(x_1, \ldots, x_n) \geq k. \tag{6.5}$$

Note that in particular this implies that if X is the zero polynomial, then any truth assignment satisfies exactly a $(1 - 2^{-r})$ fraction of the original clauses. By Lemma 6.1.2 and Lemma 6.1.3, $\mathbb{P}(X \geq \frac{\sqrt{\mathbb{E}(X^2)}}{2\sqrt{b}}) > 0$, where $b = 9^r$ and $\mathbb{E}(X^2) = \sum_{I \in \mathcal{S}} c_I^2 \geq |\mathcal{S}|$; the last inequality follows from the fact that each $|c_I|$ is a positive integer. Therefore $\mathbb{P}(X \geq \frac{\sqrt{|\mathcal{S}|}}{2 \cdot 3^r}) > 0$. Now, if $k \leq \frac{\sqrt{|\mathcal{S}|}}{2 \cdot 3^r}$ then there are $x_i \in \{-1, 1\}$ such that (6.5) holds, and there is an assignment for which the answer to Max-r-Sat$_{\text{TLB}}$ is Yes. Otherwise, $|\mathcal{S}| = O(k^2)$, and in fact even $\sum_{I \in \mathcal{S}} |c_I| \leq \sum_{I \in \mathcal{S}} c_I^2 = O(k^2)$, that is, the total number of terms of the simplified polynomial, even when counted with multiplicities, is at most $O(k^2)$.

For any fixed r, the representation of a problem instance of m clauses as a polynomial, and the simplification of this polynomial, can be performed in time $O(m)$. If the number of nonzero terms of this polynomial is larger than $4 \cdot 3^{2r} k^2$, then the answer to the problem is Yes. Otherwise, the polynomial has at most $O(k^2)$ terms and depends on at most $O(k^2)$ variables, and its maximum can be found in time $2^{O(k^2)}$.

This completes the proof of the first part of the theorem. We next establish the second part. Given the simplified polynomial X as above, define a problem in Max-Lin2$_{\text{TLB}}$ with the variables $z_1, z_2, \ldots, z_n$ as follows. For each nonzero term $c_I \prod_{i \in I} x_i$ consider the linear equation $\sum_{i \in I} z_i = b$, where $b = 0$ if c_I is positive, and $b = 1$ if c_I is negative, and either associate this equation with the weight $w_I = |c_I|$, or duplicate it $|c_I|$ times. It is easy to check that this system of equations has an assignment z_i satisfying at least $[\sum_{I \in \mathcal{S}} w_I + k]/2$ of the equations if and only if there are $x_i \in \{-1, 1\}$ so that $X(x_1, x_2, \ldots, x_n) \geq k$. This is shown by the transformation $x_i = (-1)^{z_i}$. See also [74] and [68] for a similar discussion. Since, as explained above, we may assume that $\sum_{I \in \mathcal{S}} |c_I| = O(k^2)$ (as otherwise we know that the answer to our problem is Yes), this provides the required bikernel of size $O(k^2)$ to Max-Lin2$_{\text{TLB}}$.

It remains to prove the existence of a polynomial size kernel for the original problem. One way to do that is to apply Lemma 2.4.1. Indeed, Max-Lin2$_{\text{TLB}}$ is in NP, and Max-r-Sat$_{\text{TLB}}$ is NP-complete, implying the desired result.

It is also possible to give a direct proof, which shows that the problem admits a kernel of size at most $O(k^2)$. To do so, we replace each linear equation of at most r variables by a set of 2^{r-1} clauses, so that if the variables z_i satisfy the equation, the corresponding Boolean variables $x_i = (-1)^{z_i}$ satisfy all these clauses, and if the variables z_i do not satisfy the equation, then the variables x_i above satisfy only $2^{r-1} - 1$ of the clauses. This is done as follows. Consider, first, a linear equation with exactly r variables. After renumbering the variables, if needed, a typical equation is of the form $z_1 + z_2 + \cdots + z_r = b$, where the sum is over $\mathbb{F}_2$ and $b \in \{0, 1\}$. There are exactly 2^{r-1} Boolean assignments $\delta = (\delta_1, \delta_2, \ldots, \delta_r)$ for the variables z_i that do **not** satisfy the equation. For each such assignment δ let C_δ be the clause consisting of r literals, where the literal number i is x_i if $\delta_i = 0$ and is $\overline{x_i}$ if $\delta_i = 1$. Note that if the variables $z_1, z_2, \ldots, z_r$ satisfy the above equation, then $(z_1, z_2, \ldots, z_r)$ is not one of the vectors δ considered, and hence each of the clauses C_δ constructed contains at least one satisfied literal when $x_i = (-1)^{z_i}$. Therefore, in this case all clauses are satisfied. A similar argument shows that if the variables z_i do not satisfy the equation, there will be exactly one non-satisfied clause, namely the one corresponding to the vector $\delta = (z_1, z_2, \ldots, z_r)$. The construction can be extended to equations with less than r variables. Indeed, the only property used in the transformation above is that there are exactly 2^{r-1} Boolean assignments for the variables $z_1, z_2, \ldots, z_r$ that do not satisfy the equation. If the equation has only $(1 \leq)$ $s < r$ variables, add to these variables an arbitrary set of $r - s$ of the other variables, and consider the set of all Boolean assignments to this augmented set of variables that do not satisfy the equation. Here, too, there are exactly 2^{r-1} such assignments and we can thus repeat the construction above in this case as well.

The above procedure transforms a set of W linear equations over $\mathbb{F}_2$ into a multiset of

$2^{r-1}W$ clauses. Moreover, if some truth assignment does not satisfy exactly ℓ equations, then the same assignment does not satisfy the same number, ℓ, of clauses. In particular, there is an assignment satisfying all equations but $(W-k)/2$ of them, if and only if there is an assignment satisfying all clauses but $(W-k)/2$ of them. This means that among the $m = 2^{r-1}W$ clauses, the number of satisfied ones is $m-(W-k)/2 = [(2^r-1)m+2^{r-1}k]/2^r$. This reduces an instance of Max-Lin2$_{\text{TLB}}$ with W equations and parameter k to an instance of Max-r-Sat$_{\text{TLB}}$ with $2^{r-1}W$ clauses and parameter $2^{r-1}k$. Since r is a constant, this provides the required kernel of size $O(k^2)$, completing the proof. □

Our algorithm for the problem Max-r-Sat$_{\text{TLB}}$ can be easily modified to provide, efficiently, for any given instance of m clauses to which there is a truth assignment satisfying at least $k/2^r$ clauses above the average, an assignment for the variables with this property. Indeed, the proof of Theorem 6.5.2 only requires that the variables x_i are $4r$-wise independent, and there are known constructions of polynomial size sample spaces supporting such random variables (see, e.g., [7], Chapter 16). Thus, if in the polynomial X, $\sqrt{|S|}/(2 \cdot 3^r) \geq k$, then one can find an assignment satisfying at least as many clauses as needed by going over all points in such a sample space, and if $\sqrt{|S|}/(2 \cdot 3^r) < k$, one can solve the problem by an exhaustive search.

6.6 Boolean Constraint Satisfaction Problems

The fixed-parameter tractability result on Max-r-Sat$_{\text{TLB}}$ can be easily extended to any family of Boolean r-Constraint Satisfaction Problems. Here is an outline of the argument.

Let r be a fixed positive integer, let Φ be a set of Boolean functions, each involving at most r variables, and let $\mathcal{F} = \{f_1, f_2, \ldots, f_m\}$ be a collection of Boolean functions, each being a member of Φ, and each acting on some subset of the n Boolean variables $x_1, x_2, \ldots, x_n$. The Boolean Max-r-Constraint Satisfaction Problem (corresponding to Φ), which we denote by the Max-r-CSP problem, for short, when Φ is clear from the context, is the problem of finding a truth assignment to the variables so as to maximize the total number of functions satisfied. Note that this includes, as a special case, the Max-r-Sat problem considered in the previous section, as well as many related problems. As most interesting problems of this type are NP-hard, we consider their parameterized version, where the parameter is, as before, the number of functions satisfied minus the expected value of this number. Note, in passing, that the above expected value is a tight lower bound for the problem, whenever the family Φ is closed under replacing each variable by its complement, since if we apply any Boolean function to all 2^r choices of literals whose underlying variables are any fixed set of r variables, then any truth assignment to the variables satisfies exactly the same number of these 2^r functions.

For each Boolean function f of $r(f)$ Boolean variables

$$x_{i_1}, x_{i_2}, \dots, x_{i_{r(f)}},$$

define a random variable X_f as follows. As in the discussion of the Max-r-Sat problem, suppose each variable x_{i_j} attains values in $\{-1, 1\}$. Let $V \subseteq \{-1, 1\}^{r(f)}$ denote the set of all satisfying assignments of f. Then

$$X_f(x_1, x_2, \dots, x_n) = \sum_{v=(v_1,\dots,v_{r(f)})\in V} 2^{r-r(f)}[\prod_{j=1}^{r(f)}(1 + x_{i_j}v_j) - 1].$$

This is a random variable defined over the space $\{-1, 1\}^n$ and its value at $x = (x_1, x_2, \dots, x_n)$ is $2^r - |V| \cdot 2^{r-r(f)}$ if x satisfies f, and is $-|V| \cdot 2^{r-r(f)}$ otherwise. Thus, the expectation of X_f is zero. Define now $X = \sum_{f\in\mathcal{F}} X_f$. Then the value of X at $x = (x_1, x_2, \dots, x_n)$ is precisely $2^r(s - a)$, where s is the number of the functions satisfied by the truth assignment x, and a is the average value of the number of satisfied functions. Our objective is to decide if X attains a value of at least k. As this is a polynomial of degree at most r with integer coefficients and expectation zero, we can repeat the arguments of Section 6.5 and prove that, for every fixed r, the problem is fixed-parameter tractable. Moreover, our previous arguments show that the problem admits a polynomial-size bikernel reducing it to an instance of Max-Lin2$_{\text{TLB}}$ of size $O(k^2)$, and if the specific r-CSP problem considered is NP-complete, then there is a polynomial size kernel. This is the case for most interesting choices of the family Φ.

Chapter 7

Combinatorial Approaches

In Chapter 6, we considered several constraint satisfaction problems parameterized above their tight lower bounds and exhibited the existence of polynomial kernels using Strictly Above/Below Expectation Method. The method SABEM is generic as well as powerful, and yet a problem-specific approach is still valuable to achieve a better asymptotic computational behavior. In this chapter, we discuss some of such efforts. We consider the problems Max-2-Sat and a wide special case of Max-Lin2, and present kernelzations whose sizes are smaller than those obtained using SABEM in the Chapter 6.

7.1 MAX-2-SAT

In this section we describe an alternative, more combinatorial, approach to the problem Max-r-Sat$_{\text{TLB}}$ for $r = 2$. Although this approach is somewhat more complicated than the one discussed in Chapter 6, it provides an additional insight to this special case of the problem, and allows us to obtain a kernel with a linear number of vertices for Max-2-Sat$_{\text{TLB}}$.

Semicomplete Reduction We start with a simple reduction rule that applies to any value of r. We say that a pair of distinct clauses Y and Z has a *conflict* if there is a literal $p \in Y$ such that $\overline{p} \in Z$. We say that an r-CNF formula F is *semicomplete* if the number of clauses is $m = 2^r$ and every pair of distinct clauses of F has a conflict. A semicomplete r-CNF formula is *complete* if each clause is over the same set of variables. There are r-CNF formulas that are semicomplete but not complete; consider for example $\{xy, x\overline{y}, \overline{x}z, \overline{xz}\}$. We have the following:

Lemma 7.1.1. *Every truth assignment to a semicomplete r-CNF formula satisfies exactly $2^r - 1$ clauses.*

Proof. Let S be a semicomplete r-CNF formula. To prove that no truth assignment satisfies all clauses of S we use the following simple counting argument from [75]. Observe that every clause is not satisfied by exactly 2^{n-r} truth assignments. However, each of these assignments satisfies each other clause (due to the conflicts). So, we have exactly $2^r \cdot 2^{n-r}$ truth assignments not satisfying S. But $2^r \cdot 2^{n-r} = 2^n$, the total number of truth assignments.

Now let τ be a truth assignment of S. By the above, τ does not satisfy a clause C of S. However, τ satisfies any other clause of S as any other clause has a conflict with C. □

Consider the following data reduction procedure.

Given an r-CNF formula F that contains a semicomplete subset $F' \subseteq F$, delete F' from F and consider $F \setminus F'$ instead. Let F^S denote the formula obtained from F by applying this deletion process as long as possible. We say that F^S is obtained from F by *semicomplete reduction*.

We state the following two simple observations as a lemma.

Lemma 7.1.2. *Let F be an r-CNF formula.*

1. *F^S can be obtained from F in polynomial time.*

2. $\mathrm{sat}(F) - \mathrm{sat}(F^S) = (1 - 2^{-r})(|F| - |F^S|)$.

7.1.1 Kernelization

Let F be a 2-CNF formula. A variable $x \in \mathrm{var}(F)$ is *insignificant* if for each literal y the numbers of occurrences of the two clauses xy and $\overline{x}y$ in F are the same. A variable $x \in \mathrm{var}(F)$ is *significant* if it is not insignificant. A literal is significant or insignificant if its underlying variable is significant or insignificant, respectively.

Theorem 7.1.3. *Let F be a 2-CNF formula with $F = F^S$ (i.e., F contains no semicomplete subsets) and let $k \geq 0$ be an integer. If F has more than $3k - 2$ significant variables, then* $\mathrm{sat}(F) \geq (3|F| + k)/4$.

The remainder of this section is devoted to the proof of Theorem 7.1.3 and its corollary. Let F be a 2-CNF formula with m clauses and n variables and let k be an integer. We assume that F contains no semicomplete subsets, i.e., $F = F^S$.

For a literal x let $c(x)$ denote the number of clauses in F containing x. Given a pair of literals x and y, $x \neq \overline{y}$, let $c(xy)$ be the number of occurrences of clause xy in F.

Given a clause $C \in F$ and a variable $x \in \mathrm{var}(F)$, let $\delta_C(x)$ be an indicator variable whose value is set as $\delta_C(x) = 1$ if $x \in C$, $\delta_C(x) = -1$ if $\overline{x} \in C$, and $\delta_C(x) = 0$ otherwise.

Lemma 7.1.4. *For each subset* $R = \{x_1, \dots, x_q\} \subseteq \mathrm{var}(F)$ *we have* $\mathrm{sat}(F) \geq (3m + k_R)/4$ *for*

$$k_R = \sum_{1\leq i\leq q} (c(x_i) - c(\overline{x_i})) + \sum_{1\leq i<j\leq q} (c(x_i\overline{x_j}) + c(\overline{x_i}x_j) - c(x_ix_j) - c(\overline{x_i x_j})).$$

Proof. Take a random truth assignment $\tau \in 2^{\mathrm{var}(F)}$ such that $\tau(x_i) = 1$ for all $i \in \{1, \dots, q\}$ and $\mathbb{P}(\tau(x) = 1) = 0.5$ for all $x \in \mathrm{var}(F) \setminus R$. A simple case analysis yields that the probability that a clause $C \in F$ is satisfied by τ is given by

$$\mathbb{P}(\tau \text{ satisfies } C) = 1 - \frac{1}{4} \prod_{1\leq i\leq q} (1 - \delta_C(x_i)).$$

Observe that for any clause C and any three distinct variables x, y, z we have $\delta_C(x)\delta_C(y)\delta_C(z) = 0$ as $\mathrm{var}(C)$ contains exactly two variables. Hence we can determine the expected number of clauses satisfied by τ as follows.

$$\begin{aligned}
\mathbb{E}(\mathrm{sat}(\tau, F)) &= \sum_{C\in F} \mathbb{P}[\ \tau \text{ satisfies } C\ ] \\
&= \sum_{C\in F} \{1 - \frac{1}{4} \prod_{1\leq i\leq q} (1 - \delta_C(x_i))\} \\
&= \frac{3}{4}m + \frac{1}{4} \sum_{C\in F} \{ \sum_{1\leq i\leq q} \delta_C(x_i) - \sum_{1\leq i<j\leq q} \delta_C(x_i)\delta_C(x_j)\} \\
&= \frac{3}{4}m + \frac{1}{4} \{ \sum_{1\leq i\leq q} \sum_{C\in F} \delta_C(x_i) - \sum_{1\leq i<j\leq q} \sum_{C\in F} \delta_C(x_i)\delta_C(x_j)\} \\
&= \frac{3}{4}m + \frac{1}{4}k_R.
\end{aligned}$$

□

It is noteworthy that $\mathbb{P}(\tau \text{ satisfies } C) = 1 - \frac{1}{4}\prod_{1\leq i\leq q}(1 - \delta_C(x_i))$ in the proof of Lemma 7.1.4 is similar to a term of X defined in section 6.5. The term $1 - \prod_{x_i\in\mathrm{var}(C)}(1 + \varepsilon_i x_i)$ of X returns a fixed value on C for a given (*fully determined*) truth assignment, depending on whether C is satisfied or not. Similarly, the term $1 - \frac{1}{4}\prod_{1\leq i\leq q}(1 - \delta_C(x_i))$ returns a probability of C being satisfied for a given (*partially determined*) random truth assignment. The benefit of having a probabilistic form of X is that we now have a way to ignore a large number of variables, e.g., $V \setminus R$ in Lemma 7.1.4, instead of searching for fully determined truth assignment so as to compute X. For the case $r = 2$, this probabilistic form of X can be immediately interpreted in a graph-theoretic language as will be shown below.

Due to Lemma 7.1.4, the task is now reduced into finding a subset R of variables with $k_R \geq k$. These are variables which form the deterministic part of a partially random truth assignment. Using a notion of *switch* defined later, we replace F by an equivalent formula in which every variable of R is set to 1. To find R we use a graph-theoretical approach introducing an auxiliary weighted graph in which we seek an induced subgraph of weight at least k. In particular, we note that an 'independent' structure of an induced subgraph ensures its weight to be above a certain bound growing with the size of the induced subgraph. This means that if (F, k) is a No-instance, we do not have a large 'independent' structure. Combining Tutte-Berge formula with this fact reveals an upper bound on the number of vertices in the auxiliary weighted graph.

We construct an *auxiliary graph* $G = (V, E)$ from F by letting $V = \mathrm{var}(F)$ and $xy \in E$ if and only if there exists a clause $C \in F$ with $\mathrm{var}(C) = \{x, y\}$ (equivalently, $c(x\overline{y}) + c(\overline{x}y) + c(xy) + c(\overline{xy}) \geq 1$).

We assign a *weight* to each vertex x and edge xy of $G = (V, E)$:

$$\begin{aligned} w(x) &:= \sum_{C \in F} \delta_C(x) = c(x) - c(\overline{x}), \\ w(xy) &:= -\sum_{C \in F} \delta_C(x)\delta_C(y) = c(x\overline{y}) + c(\overline{x}y) - c(xy) - c(\overline{xy}). \end{aligned}$$

For subsets $U \subseteq V$ and $H \subseteq E$, let $w(U) = \sum_{x \in U} w(x)$ and $w(H) = \sum_{xy \in H} w(xy)$. The *weight* $w(Q)$ of a subgraph $Q = (U, H)$ is $w(U) + w(H)$. Let G^0 be the graph obtained from G by removing all edges of weight zero.

Lemma 7.1.5. *A variable $x \in \mathrm{var}(F)$ is insignificant if and only if x is an isolated vertex in G^0 and $w(x) = 0$.*

Proof. Suppose $x \in \mathrm{var}(F)$ is insignificant. Choose an edge $xy \in E$ (this is possible since by construction G has no isolated vertices). Since x is insignificant, $c(x\overline{y}) = c(\overline{xy})$ and $c(xy) = c(\overline{x}y)$ and thus $w(xy) = 0$. Therefore the edge xy does not appear in G^0 and x is isolated in G^0. Observe that we have $c(x) = c(\overline{x})$, which implies $w(x) = 0$.

Suppose $x \in \mathrm{var}(F)$ is an isolated vertex of G^0 and $w(x) = 0$. Since G has no isolated vertices, we have $w(xy) = 0$ for all $xy \in E$. In order to derive a contradiction, let us suppose x is a significant variable of F. Consequently there is (i) either a clause $xy \in F$ such that $c(xy) > c(\overline{x}y)$, or (ii) there is a clause $\overline{x}y \in F$ such that $c(\overline{x}y) > c(xy)$. We consider case (i) only, case (ii) can be treated analogously. With $w(xy) = 0$, we have $c(x\overline{y}) > c(\overline{xy})$, and thus $x\overline{y} \in F$.

Now the condition $w(x) = c(x) - c(\overline{x}) = 0$ implies the existence of an edge $xz \in E$ with $z \neq y$ such that for some $z' \in \{z, \overline{z}\}$ we have $\overline{x}z' \in F$ and $c(\overline{x}z') > c(xz')$. Without loss of generality, assume that $z' = z$. Since $w(xz) = 0$, we have $\overline{xz} \in F$. However, the four clauses xy, $x\overline{y}$, $\overline{x}z$, $\overline{xz}$ in F form a semicomplete 2-CNF formula, which contradicts our assumption that $F = F^S$. Hence x is indeed an insignificant variable. □

For a set $X \subseteq \mathrm{var}(F)$ we let F_X denote the 2-CNF formula obtained from F by replacing x with $\overline{x}$ and $\overline{x}$ with x for each $x \in X$. We say that F_X is obtained from F by *switching* X.

The following lemma follows immediately from the definitions of switch and weights.

Lemma 7.1.6. *The auxiliary graph G_X corresponding to F_X can be obtained from $G = (V, E)$ by reversing the signs of the weights of all vertices in X and all edges between X and $V \setminus X$. Moreover,* $\mathrm{sat}(F) = \mathrm{sat}(F_X)$.

To distinguish between weights in G and G_X, we use $w_X(.)$ for weights of G_X. Similarly, we use $c_X(.)$ for F_X.

It is sometimes convenient to stress that the set X we are switching induces a subgraph. We can *switch an induced graph* Q by switching all the vertices of Q. Observe that by switching an induced graph Q, we reverse the signs of weights on all vertices of Q and all edges incident with exactly one vertex of Q, but the sign of each edge within Q remains unchanged. This property will play a major role to show that a certain structure meets the condition of the following lemma.

Lemma 7.1.7. *If there exist a set $X \subset V(G^0)$ and an induced subgraph $Q = (U, H)$ of G^0 with $w_X(Q) \geq k$, then* $\mathrm{sat}(F) \geq (3m + k)/4$.

Proof. We consider $U = \{x_1, \ldots, x_q\}$ as a subset of $\mathrm{var}(F_X)$. By Lemmas 7.1.4 and 7.1.6, $\mathrm{sat}(F) = \mathrm{sat}(F_X) \geq (3m + k_U)/4$, where

$$\begin{aligned} k_U &= \sum_{i=1}^{q}(c_X(x_i) - c_X(\overline{x_i})) + \sum_{1 \leq i < j \leq q}(c_X(x_i\overline{x_j}) + c_X(\overline{x_i}x_j) - c_X(x_ix_j) - c_X(\overline{x_i}\overline{x_j})) \\ &= \sum_{i=1}^{q} w_X(x_i) + \sum_{1 \leq i < j \leq q} w_X(x_ix_j) \quad = \quad w_X(Q) \quad \geq \quad k. \qquad \square \end{aligned}$$

To apply Lemma 7.1.7 in the proof of Theorem 7.1.3, we will focus on a special case of induced subgraphs of G^0. For a set $U \subseteq V(G^0)$, let $G^0[U]$ denote the subgraph of G^0 induced by U. We call $G^0[U]$ an *induced star* with *center* x if x is a vertex of G^0, I is an independent set in the subgraph of G^0 induced by the neighbors of x and $U = \{x\} \cup I$. We are interested in the induced star due to the following property.

Lemma 7.1.8. *Let x be the center of an induced star $Q = G^0[U]$ and let $I = U \setminus \{x\}$. Then there is a set $X \subseteq U$ such that $w_X(Q) \geq |I|$.*

Proof. Let H be the set of edges of Q. We may assume that $w(xy) \geq 0$ for each $y \in I$ since otherwise we can switch y, and $w(xy)$ is integral. By a *random switch* of Q, we mean a switch of every vertex of Q with probability 0.5. Take a random switch R of Q.

Then we have $\mathbb{E}(w_R(z)) = 0$ for all $z \in U$. Note that the sign of each edge in H remains positive. Hence we have $\mathbb{E}(w_R(Q)) = w(H) \geq |I|$ and thus there exists a set $X \subseteq U$ for which $w_X(Q) \geq |I|$. □

If we are given more than one induced star, a sequence of random switches gives us a similar result.

Lemma 7.1.9. *Let $Q_1 = (U_1, H_1), \ldots, Q_m = (U_m, H_m)$ be a collection of vertex-disjoint induced stars of G^0 with centers $x_1, \ldots, x_m$, let $U = \bigcup_{i=1}^{m} U_i$, and let $Q = G^0[U]$. Then there is a set $X \subseteq U$ such that $w_X(Q) \geq \sum_{i=1}^{m} |I_i|$, where $I_i = U_i \setminus \{x_i\}$, $i = 1, \ldots, m$.*

Proof. As in the proof of Lemma 7.1.8, we may assume that all the edges of H_i have positive weights. Let H be the set of edges of Q. By a *random switch* of Q, we mean a sequence of switches of $Q_1, \ldots, Q_m$ each with probability 0.5. Take a random switch R of Q. Then we have $\mathbb{E}(w_R(x)) = 0$ for all $x \in U$. Moreover, for the subgraph Q of G^0_R, it holds that $\mathbb{E}(w_R(xy)) = 0$ for all $xy \in H \setminus \bigcup_{i=1}^{m} H_i$ since each choice of $w_R(xy) \geq 0$ and $w_R(xy) \leq 0$ is equally likely. By linearity of expectation and Lemma 7.1.8, we have $\mathbb{E}(w_R(Q)) = w(\bigcup_{i=1}^{m} H_i) \geq \sum_{i=1}^{m} |I_i|$ and thus there exists a set $X \subseteq U$ for which $w_X(Q) \geq \sum_{i=1}^{m} |I_i|$. □

Note that we can derandomize the procedures suggested in the proofs of Lemma 7.1.8 and 7.1.9 using the standard technique of conditional expectation [7].

We are now in the position to complete the proof of Theorem 7.1.3.

Suppose that (F, k) is a no-instance, i.e., $\text{sat}(F) < (3m + k)/4$. Notice that a matching can be viewed as a collection of induced stars of G^0 for which $|I_i| = 1$. It follows by Lemmas 7.1.7 and 7.1.9 that G^0 has no matching of size k. The Tutte-Berge formula [15, 26] states that the size of a maximum matching in G^0 equals

$$\min_{S \subseteq V(G^0)} \frac{1}{2}\{|V(G^0)| + |S| - oc(G^0 - S)\}$$

where $oc(G^0 - S)$ is the number of odd components (connected components with an odd number of vertices) in $G^0 - S$. Hence there is a set $S \subseteq V(G^0)$ such that $|V(G^0)| + |S| - oc(G^0 - S) < 2k$. It follows that

$$|V(G^0)| \leq oc(G^0 - S) - |S| + 2k - 1. \tag{7.1}$$

We will now classify odd components in $G^0 - S$. One obvious type of odd components is an isolated vertex in G^0 of weight zero, which corresponds to an insignificant variable by Lemma 7.1.5. All the other odd components can be categorized into one of the following two types:

1. Let $Q_1, \dots, Q_L$ be the odd components of $G^0 - S$ such that for all $1 \leq i \leq L$ we have $|Q_i| = 1$ and Q_i is a significant variable.

2. Let $Q'_1, \dots, Q'_{L'}$ be the odd components of $G^0 - S$ such that for all $1 \leq i \leq L'$ we have $|Q'_i| > 1$.

We construct a collection of induced stars as follows. From each of $Q'_1, \dots, Q'_{L'}$ we choose an edge, which is an induced star with $|I| = 1$. Let us consider $Q_1, \dots, Q_L$. Each vertex Q_i is adjacent to at least one vertex of S. Thus, we can partition $Q_1, \dots, Q_L$ into $|S|$ sets, some of them possibly empty, such that each partite set forms an independent set in which every vertex is adjacent to the corresponding vertex x_i of S. Each partite set, together with x_i, forms an induced star. Now observe that we have a collection of induced stars and the total number of edges equals $L + L'$. If $L + L' \geq k$, Lemma 7.1.9 implies that for some set X of vertices from the odd components $w_X(Q) \geq k$, which is impossible by Lemma 7.1.7. Hence $L + L' \leq k - 1$.

Therefore, $oc(G^0 - S) - n' = L + L' \leq k - 1$, where n' is the number of insignificant variables. By (7.2), we have $|V(G^0)| - n' \leq k - 1 - |S| + 2k - 1 \leq 3k - 2$. It remains to observe that $|V(G^0)| - n'$ equals the number of significant variables of F. This completes the proof of Theorem 7.1.3.

Corollary 7.1.10. *The problem* Max-2-Sat$_{\text{TLB}}$ *admits a (polynomial time) reduction to a problem kernel with at most* $3k - 1$ *variables.*

Proof. Consider an instance (F, k) of the problem. First we apply the semicomplete reduction and obtain (in polynomial time) an instance (F', k) with $F' = F^S$. We determine (again in polynomial time) the set S' of significant variables of F'. If $|S'| > 3k - 2$ then (F', k) is a yes-instance by Theorem 7.1.3, and consequently (F, k) is a yes-instance by Lemma 7.1.2. Assume now that $|S'| \leq 3k - 2$.

Let z be a new variable not occurring in F. Since $F' = F^S$, no clause contains two insignificant variables and, thus, each insignificant variable can be replaced by z without changing the solution to (F', k). Let us denote the modified F' by F''; F'' has at most $3k - 1$ variables.

Let p be the number of clauses in F''. Observe that we can find a truth assignment satisfying the maximum number of clauses of F'' in time $O(p8^k)$. Thus, if $p > 8^k$, we can find the optimal truth assignment in the polynomial time $O(p^2) = O(m^2)$. Thus, we may assume that F'' has at most 8^k clauses. Therefore, F'' is a kernel of the Max-2-Sat$_{\text{TLB}}$ problem. □

7.2 A Family of Special Cases of Max-Lin2

In the problem Max-Lin2, notice that maximizing the total weight of satisfied equations is equivalent to maximizing the *excess*, which is the total weight of satisfied equations minus the total weight of falsified equations. We investigate lower bounds for the maximum excess. Using an algebraic approach, we prove the following main result: Let $Az = b$ be a Max Lin system such that rank$A = n$ and no pair of equations has the same left-hand side, let $w_{\min}$ be the minimum weight of an equation in $Az = b$, and let $k \geq 2$. If $k \leq m \leq 2^{n/(k-1)} - 2$, then the maximum excess of $Az = b$ is at least $k \cdot w_{\min}$. Moreover, we can find an assignment that achieves an excess of at least $k \cdot w_{\min}$ in time $m^{O(1)}$.

Due to the data reduction rules 4 and 5 presented in Section 6.3, we may assume that no two equations in $Az = b$ have the same left-hand side and $n =$ rankA. Using our maximum excess results, we prove that, under these assumptions, (a) Max-Lin2$_{\text{TLB}}$ is fixed-parameter tractable if $m \leq 2^{p(n)}$ for an arbitrary fixed function $p(n) = o(n)$, and (b) Max-Lin2$_{\text{TLB}}$ has a polynomial-size kernel if $m \leq 2^{n^a}$ for an arbitrary $a < 1$. In addition, we prove that Max-Lin2$_{\text{TLB}}$ is in XP (thus, Max-Lin2$_{\text{TLB}}$ is polynomial-time solvable for every fixed k), and, moreover, it is in W[P].

Recall that Max-r-Lin2$_{\text{TLB}}$ is a special case of Max-Lin2$_{\text{TLB}}$, where each equation has at most r variables. Using our maximum excess results, we prove that for each fixed $r \geq 2$ Max-r-Lin2$_{\text{TLB}}$ has a kernel with $O(k \log k)$ variables and, thus, it can be solved in time $2^{O(k \log k)} + m^{O(1)}$. This improves a kernel with $O(k^2)$ variables for Max-r-Lin2$_{\text{TLB}}$ obtained in Section 6.3 using the generic method SABEM. Similarly, we prove that for each $r \geq 2$ Max-r-Sat$_{\text{TLB}}$ has a kernel with $O(k \log k)$ variables and it can be solved in time $2^{O(k \log k)} + m^{O(1)}$ improving a kernel with $O(k^2)$ variables for Max-r-Sat$_{\text{TLB}}$ obtained in Section 6.5. Note that while the kernels with $O(k^2)$ variables were obtained using a probabilistic approach, our results are obtained using an algebraic approach.

In Fourier analysis, the Boolean domain is often assumed to be $\{-1, +1\}^n$ rather than more usual $\{0, 1\}^n$ and we will follow this assumption in our work. Here we use the following well-known and easy to prove fact [97] that each function $f : \{-1, +1\}^n \to \mathbb{R}$ can be uniquely written as

$$f(x) = \sum_{S \subseteq [n]} c_S \prod_{i \in S} x_i, \tag{7.2}$$

where $[n] = \{1, 2, \ldots, n\}$ and each c_S is a real. Formula (7.2) is the Fourier expansion f, c_S are the Fourier coefficients of f, and the monomials $\prod_{i \in S} x_i$ form an orthogonal basis of (7.2) (thus, the monomials are often written as $\chi_S(x)$ but we will use only $\prod_{i \in S} x_i$ as it is more transparent).

7.2.1 Results on Maximum Excess

We consider the two reduction rules 4 and 5 from Section 6.3. These rules are of interest due to Lemma 7.2.1.

Reminder of Reduction Rule 4 *Let* $t = \mathrm{rank}A$ *and let columns* $a^{i_1}, \ldots, a^{i_t}$ *of* A *be linearly independent. Then delete all variables not in* $\{z_{i_1}, \ldots, z_{i_t}\}$ *from the equations of* $Az = b$*.*

Reminder of Reduction Rule 5 *If we have, for a subset* S *of* $[n]$*, an equation* $\sum_{i \in S} z_i = b'$ *with weight* w'*, and an equation* $\sum_{i \in S} z_i = b''$ *with weight* w''*, then we replace this pair by one of these equations with weight* $w' + w''$ *if* $b' = b''$ *and, otherwise, by the equation whose weight is bigger, modifying its new weight to be the difference of the two old ones. If the resulting weight is 0, we delete the equation from the system.*

Lemma 7.2.1. *Let* $A'z' = b'$ *be obtained from* $Az = b$ *by applying Rule 4 or 5. Then the maximum excess of* $A'z' = b'$ *is equal to the maximum excess of* $Az = b$*. Moreover,* $A'z' = b'$ *can be obtained from* $Az = b$ *in time polynomial in* n *and* m*.*

To see the validity of Rule 4, consider an independent set I of columns of A of cardinality rank$A$ and a column $a^j \notin I$. Observe that $a^j = \sum_{i \in I'} a^i$, where $I' \subseteq I$. Consider an assignment $z = z^0$. If $z^0_j = 1$ then for each $i \in I' \cup \{j\}$ replace z^0_i by $z^0_i + 1$. The new assignment satisfies exactly the same equations as the initial assignment. Thus, we may assume that $z_j = 0$ and remove z_j from the system. If we cannot change a weighted system $Az = b$ using Rules 4 and 5, we call it *irreducible*.

Consider the following algorithm that tries to maximize the total weight of satisfied equations of $Az = b$. We assume that, in the beginning, no equation or variable in $Az = b$ is marked.

Algorithm $\mathcal{H}$

While the system $Az = b$ is nonempty do the following:

1. Choose an arbitrary equation $\sum_{i \in S} z_i = b$ and mark z_l, where $l = \min\{i : i \in S\}$.

2. Mark this equation and delete it from the system.

3. Replace every equation $\sum_{i \in S'} z_i = b'$ in the system containing z_l by $\sum_{i \in S} z_i + \sum_{i \in S'} z_i = b + b'$. (The weight of the equation is unchanged.)

4. Apply Reduction Rule 5 to the system.

Note that algorithm $\mathcal{H}$ replaces $Az = b$ with an *equivalent* system under the assumption that the marked equations are satisfied; that is, for every assignment of values to the variables $z_1, \ldots, z_n$ that satisfies the marked equations, both systems have the same excess.

The *maximum $\mathcal{H}$-excess* of $Az = b$ is the maximum possible total weight of equations marked by $\mathcal{H}$ for $Az = b$ taken over all possible choices in Step 1 of $\mathcal{H}$.

Lemma 7.2.2. *The maximum excess of $Az = b$ equals its maximum $\mathcal{H}$-excess.*

Proof. We first prove that the maximum excess of $Az = b$ is not smaller than its maximum $\mathcal{H}$-excess.

Let K be the set of equations marked by $\mathcal{H}$. A method first described in [39] can find an assignment of values to the variables such that the equations in K are satisfied and, in the remainder of the system, the total weight of satisfied equations is not smaller than the total weight of falsified equations.

For the sake of completeness, we repeat the description here. By construction, for any assignment that satisfies all the marked equations, exactly half of the non-marked equations are satisfied. Therefore it suffices to find an assignment to the variables such that all marked equations are satisfied. This is possible if we find an assignment that satisfies the last marked equation, then find an assignment satisfying the equation marked before the last, etc. Indeed, the equation marked before the last contains a (marked) variable z_l not appearing in the last equation, etc. This proves the first part of our lemma.

Now we prove that the maximum $\mathcal{H}$-excess of $Az = b$ is not smaller than its maximum excess. Let $z = (z_1, \ldots, z_n)$ be an assignment that achieves the maximum excess, t. Observe that if at each iteration of $\mathcal{H}$ we mark an equation that is satisfied by z, then $\mathcal{H}$ will mark equations of total weight t. □

Remark 7.2.3. It follows from Lemma 7.2.2 that the maximum excess of a (nonempty) irreducible system $Az = b$ with smallest weight $w_{\min}$ is at least $w_{\min}$. If all weights are integral, then the maximum excess of $Az = b$ is at least 1.

Clearly, the total weight of equations marked by $\mathcal{H}$ depends on the choice of equations to mark in Step 1. Below we consider one such choice based on the following theorem. The theorem allows us to find a set of equations such that we can mark each equation in the set in successive iterations of $\mathcal{H}$. This means we can run $\mathcal{H}$ a guaranteed number of times, which we can use to get a lower bound on the $\mathcal{H}$-excess.

Theorem 7.2.4. *Let M be a set in $\mathbb{F}_2^n$ such that M contains a basis of $\mathbb{F}_2^n$, the zero vector is in M and $|M| < 2^n$. If k is a positive integer and $k + 1 \leq |M| \leq 2^{n/k}$ then, in time $|M|^{O(1)}$, we can find a subset K of M of $k + 1$ vectors such that no sum of two or more vectors of K is in M.*

Proof. We first consider the case when $k = 1$. Since $|M| < 2^n$ and the zero vector is in M, there is a non-zero vector $v \notin M$. Since M contains a basis for $\mathbb{F}_2^n$, v can be written as a sum of vectors in M and consider such a sum with the minimum number of summands: $v = u_1 + \cdots + u_\ell$, $\ell \geq 2$. Since $u_1 + u_2 \notin M$, we may set $K = \{u_1, u_2\}$. We can find such a set K in polynomial time by looking at every pair in $M \times M$.

We now assume that $k > 1$. Since $k + 1 \leq |M| \leq 2^{n/k}$ we have $n \geq k + 1$.

We proceed with a greedy algorithm that tries to find K. Suppose we have a set $L = \{a_1, \ldots, a_l\}$ of vectors in M, $l \leq k$, such that no sum of two or more elements of L is in M. We can extend this set to a basis, so $a_1 = (1, 0, 0, \ldots, 0)$, $a_2 = (0, 1, 0, \ldots, 0)$ and so on. For every $a \in M \backslash L$ we check whether $M \backslash \{a_1, \ldots, a_l, a\}$ has an element that agrees with a in all co-ordinates $l + 1, \ldots, n$. If no such element exists, then we add a to the set L, as no element in M can be expressed as a sum of a and a subset of L.

If our greedy algorithm finds a set L of size at least $k + 1$, we are done and L is our set K. Otherwise, we have stopped at $l \leq k$. In this case, we do the next iteration as follows. Recall that L is part of a basis of M such that $a_1 = (1, 0, 0, \ldots, 0)$, $a_2 = (0, 1, 0, \ldots, 0), \ldots$. We create a new set M' in $\mathbb{F}_2^{n'}$, where $n' = n - l$. We do this[1] by removing the first l co-ordinates from M, and then identifying together any vectors that agree in the remaining n' co-ordinates. We are in effect identifying together any vectors that only differ by a sum of some elements in L. It follows that every element of M' was created by identifying together at least two elements of M, since otherwise we would have had an element in $M \backslash L$ that should have been added to L by our greedy algorithm. Therefore it follows that $|M'| \leq |M|/2 \leq 2^{n/k - 1}$. From this inequality and the fact that $n' \geq n - k$, we get that $|M'| \leq 2^{n'/k}$. It also follows by construction of M' that M' has a basis for $\mathbb{F}_2^{n'}$, and that the zero vector is in M'. (Thus, we have $|M'| \geq n' + 1$.) If $n' \geq k + 1$ we complete this iteration by running the algorithm on the set M' as in the first iteration. Otherwise ($n' \leq k$), the algorithm stops.

Since each iteration of the algorithm decreases n', the algorithm terminates. Now we prove that at some iteration, the algorithm will actually find a set K of $k + 1$ vectors. To show this it suffices to prove that we will never reach the point when $n' \leq k$. Suppose this is not true and we obtained $n' \leq k$. Observe that $n' \geq 1$ (before that we had $n' \geq k + 1$ and we decreased n' by at most k) and $|M'| \geq n' + 1$. Since $|M'| \leq 2^{n'/k}$, we have $n' + 1 \leq 2^{n'/k}$, which is impossible due to $n' \leq k$ unless $n' = 1$ and $k = 1$, a contradiction with the assumption that $k > 1$.

It is easy to check that the running time of the algorithm is polynomial in $|M|$. □

Remark 7.2.5. It is much easier to prove a non-constructive version of the above result.

[1] For the reader familiar with vector space terminology: $\mathbb{F}_2^{n'}$ is $\mathbb{F}_2^n$ modulo span(L), the subspace of $\mathbb{F}_2^n$ spanned by L, and M' is the image of M in $\mathbb{F}_2^{n'}$.

In fact we can give a non-constructive proof that $k + 1 \le |M| \le 2^{n/k}$ can be replaced by $2k < |M| < 2^{n/k}((k-1)!)^{1/k}$. We will extend our proof above for the case $k = 1$. We may assume that $k \ge 2$. Observe that the number of vectors of $\mathbb{F}_2^n$ that can be expressed as the sum of at most k vectors of M is at most

$$\binom{|M|}{k} + \binom{|M|}{k-1} + \cdots + \binom{|M|}{1} + 1 \le |M|^k/(k-1)! \text{ for } |M| > 2k.$$

Since $|M| < 2^{n/k}((k-1)!)^{1/k}$ we have $|\mathbb{F}_2^n| > |M|^k/(k-1)!$ and, thus, at least for one vector a of $\mathbb{F}_2^n$ we have $a = m_1 + \cdots + m_\ell$, where ℓ is minimum and $\ell > k$. Note that, by the minimality of ℓ, no sum of two or more summands of the sum for a is in M and all summands are distinct. Thus, we can set $K = \{m_1, \ldots, m_{k+1}\}$.

Theorem 7.2.6. *Let $Az = b$ be an irreducible system, let $w_{\min}$ be the minimum weight of an equation in $Az = b$, and let $k \ge 2$. If $k \le m \le 2^{n/(k-1)} - 2$, then the maximum excess of $Az = b$ is at least $k \cdot w_{\min}$. Moreover, we can find an assignment that achieves an excess of at least $k \cdot w_{\min}$ in time $m^{O(1)}$.*

Proof. Consider a set M of vectors in $\mathbb{F}_2^n$ corresponding to equations in $Az = b$ as follows: for each $\sum_{i \in S} z_i = b_S$ in $Az = b$, the vector $v = (v_1, \ldots, v_n) \in M$, where $v_i = 1$ if $i \in S$ and $v_i = 0$, otherwise. Add the zero vector to M. As $Az = b$ is reduced by Rule 4 and $k \le m \le 2^{n/(k-1)} - 2$, we have that M contains a basis for $\mathbb{F}_2^n$ and $k \le |M| \le 2^{n/(k-1)} - 1$. Therefore, using Theorem 7.2.4 we can find a set K of k vectors such that no sum of two or more vectors in K belongs to M.

Now run Algorithm $\mathcal{H}$ choosing at each Step 1 an equation of $Az = b$ corresponding to a member of K, then equations picked at random until the algorithm terminates. Algorithm $\mathcal{H}$ will run at least k iterations as no equation corresponding to a vector in K will be deleted before it has been marked. Indeed, suppose that this is not true. Then there are vectors $w \in K$ and $v \in M$ and a pair of nonintersecting subsets K' and K'' of $K \setminus \{v, w\}$ such that $w + \sum_{u \in K'} u = v + \sum_{u \in K''} u$. Thus, $v = w + \sum_{u \in K' \cup K''} u$, a contradiction with the definition of K.

In fact, the above argument shows that no equation of $Az = b$ corresponding to a member of K will change its weight during the first k iterations of $\mathcal{H}$. Thus, by Lemma 7.2.2, the maximum excess of $Az = b$ is at least $k \cdot w_{\min}$. It remains to observe that we can once again use the algorithm given in the proof of Lemma 7.2.2 to find an assignment that gives an excess of at least $k \cdot w_{\min}$. □

We now provide a useful association between weighted systems of linear equations on $\mathbb{F}_2^n$ and Fourier expansions of functions $f : \{-1, +1\} \to \mathbb{R}$. Let us rewrite (7.2), the

Fourier expansion of such a function, as

$$f(x) = c_\emptyset + \sum_{S\in\mathcal{F}} c_S \prod_{i\in S} x_i, \tag{7.3}$$

where $\mathcal{F} = \{\emptyset \neq S \subseteq [n] : \ c_S \neq 0\}$.

Now associate the polynomial $\sum_{S\in\mathcal{F}} c_S \prod_{i\in S} x_i$ in (7.3) with a weighted system $Az = b$ of linear equations on $\mathbb{F}_2^n$: for each $S \in \mathcal{F}$, we have an equation $\sum_{i\in S} z_i = b_S$ with weight $|c_S|$, where $b_S = 0$ if c_S is positive and $b_S = 1$, otherwise. Conversely, suppose we have a system $Az = b$ of linear equations on $\mathbb{F}_2^n$ in which each equation $\sum_{i\in S} z_i = b_S$ is assigned a weight $w_S > 0$ and no pair of equations have the same left-hand side. This system can be associated with the polynomial $\sum_{S\in\mathcal{F}} c_S \prod_{i\in S} x_i$, where $c_S = w_S$, if $b_S = 0$, and $c_S = -w_S$, otherwise. The above associations provide a bijection between Fourier expansions of functions $f : \{-1, +1\} \to \mathbb{R}$ with $c_\emptyset = 0$ and weighted systems of linear equations on $\mathbb{F}_2^n$. This bijection is of interest due to the following:

Proposition 7.2.7. *An assignment $z^{(0)} = (z_1^{(0)}, \ldots, z_n^{(0)})$ of values to the variables of $Az = b$ maximizes the total weight of satisfied equations of $Az = b$ if and only if $x^{(0)} = ((-1)^{z_1^{(0)}}, \ldots, (-1)^{z_n^{(0)}})$ maximizes $f(x)$. Moreover,* $\max_{x\in\{-1,+1\}^n} f(x) - c_\emptyset$ *equals the maximum excess of $Az = b$.*

Proof. The claims of this lemma easily follow from the fact that an equation $\sum_{i\in S} z_i = 0$ is satisfied if and only if $\prod_{i\in S} x_i > 0$, where $x_i = (-1)^{z_i}$. □

7.2.2 Corollaries

This section contains a collection of corollaries of Theorem 7.2.6 establishing parameterized complexity of special cases of Max-Lin2$_{\text{TLB}}$, of Max-r-Sat$_{\text{TLB}}$, and of a wide class of constraint satisfaction problems. In addition, we will prove that Max-Lin2$_{\text{TLB}}$ is in X[P] and obtain a sharp lower bound on the maximum of a pseudo-boolean function.

Parameterized Complexity of Max-Lin2$_{\text{TLB}}$

Corollary 7.2.8. *Let $p(n)$ be a fixed function such that $p(n) = o(n)$. If $m \leq 2^{p(n)}$ then* Max-Lin2$_{\text{TLB}}$ *is fixed-parameter tractable. Moreover, a satisfying assignment can be found in time $g(k)m^{O(1)}$ for some computable function g.*

Proof. We may assume that $m \geq n > k > 1$. Observe that $m \leq 2^{n/k}$ implies $m \leq 2^{n/(k-1)} - 2$. Thus, by Theorem 7.2.6, if $p(n) \leq n/k$, the answer to Max-Lin2$_{\text{TLB}}$ is yes, and there is a polynomial algorithm to find a suitable assignment. Otherwise, $n \leq f(k)$ for some function dependent on k only and Max-Lin2$_{\text{TLB}}$ can be solved in time $2^{f(k)}m^{O(1)}$ by checking every possible assignmen. □

Let ρ_i be the number of equations in $Az = b$ containing z_i, $i = 1, \ldots, n$. Let $\rho = \max_{i \in [n]} \rho_i$ and let r be the maximum number of variables in an equation of $Az = b$. Crowston et al. [39] proved that Max-Lin2$_{\text{TLB}}$ is fixed-parameter tractable if either $r \leq r(n)$ for some fixed function $r(n) = o(n)$ or $\rho \leq \rho(m)$ for some fixed function $\rho(m) = o(m)$.

For a given $r = r(n)$, we have $m \leq \sum_{i=1}^{r} \binom{n}{i}$. By Corollary 23.6 in [77], $m \leq 2^{nH(r/n)}$, where $H(y) = -y \log_2 y - (1-y)\log_2(1-y)$, the entropy of y. It is easy to see that if $y = o(n)/n$, then $H(y) = o(n)/n$. Hence, if $r(n) = o(n)$, then $m \leq 2^{o(n)}$. By Corollary 23.5 in [77] (this result was first proved by Kleitman et al. [80]), for a given $\rho = \rho(m)$ we have $m \leq 2^{nH(\rho/m)}$. Therefore, if $\rho(m) = o(m)$ then $m \leq 2^{n \cdot o(m)/m}$ and, thus, $m \leq 2^{o(n)}$ (as $n \leq m$, if $n \to \infty$ then $m \to \infty$ and $o(m)/m \to 0$). Thus, both results of Crowston et al. [39] follow from corollary 7.2.8.

Similarly to Corollary 7.2.8 it is easy to prove the following:

Corollary 7.2.9. *Let $0 < a < 1$ be a constant. If $m < 2^{O(n^a)}$ then* Max-Lin2$_{\text{TLB}}$ *has a kernel with $O(k^{1/(1-a)})$ variables.*

By Corollary 7.2.8 it is easy to show that Max-Lin2$_{\text{TLB}}$ is in XP.

Proposition 7.2.10. Max-Lin2$_{\text{TLB}}$ *can be solved in time $O(m^{k+O(1)})$.*

Proof. We may again assume $m \geq n > k > 1$. As in the proof of Corollary 7.2.8, if $m \leq 2^{n/k}$ then the answer to Max-Lin2$_{\text{TLB}}$ is yes and a solution can be found in time $m^{O(1)}$. Otherwise, $2^n < m^k$ and Max-Lin2$_{\text{TLB}}$ can be solved in time $O(m^{k+2})$. □

In fact, it is possible to improve this result, as the next theorem shows.

Theorem 7.2.11. Max-Lin2$_{\text{TLB}}$ *is in W[P].*

To prove this theorem we make use of the following lemma from [55] (Lemma 3.8, p. 48). Here $k(x)$ is the value of the parameter on an instance $x \in \Sigma^*$.

Lemma 7.2.12. *A parameterized problem (Q, k) over the alphabet Σ is in W[P] if and only if there are computable functions $f, h : \mathbb{N} \to \mathbb{N}$, a polynomial $p(X)$, and a $Y \subseteq \Sigma^* \times \{0,1\}^*$ such that:*

(i) *For all $(x, y) \in \Sigma^* \times \{0,1\}^*$, it is decidable in time $f(k(x)) \cdot p(|x|)$ whether $(x, y) \in Y$.*

(ii) *For all $(x, y) \in \Sigma^* \times \{0,1\}^*$, if $(x, y) \in Y$ then $|y| = h(k(x)) \cdot \lfloor \log_2 |x| \rfloor$.*

(iii) *For every $x \in \Sigma^*$*

$$x \in Q \iff \text{there exists a } y \in \{0,1\}^* \text{such that } (x, y) \in Y.$$

Proof of Theorem 7.2.11. Recall from Lemma 7.2.2 that the maximum excess of $Az = b$ is at least k if and only if we can run algorithm $\mathcal{H}$ a number of times and get a total weight of marked equations at least k.

Suppose we are given a sequence $e_1, \ldots, e_l$ of equations to mark in each iteration of $\mathcal{H}$. We can, at the i'th iteration of $\mathcal{H}$, mark equation e_i as long as e_i is still in the system. If we are able to mark all the equations $e_1, \ldots e_l$, we can then check that the total weight of these marked equations is at least k. If it is, then we know we have a YES-instance. Conversely, if the system has a maximum excess of at least k, then there will be some sequence $e_1, \ldots, e_l$ that gives us a total weight of marked equations at least k. Furthermore, by integrality of the weights, we may assume that $l \leq k$. We use this idea to construct a set Y that satisfies the conditions of Lemma 7.2.12.

Firstly we show that a sequence of $l \leq k$ equations can be encoded as a string $y \in \{0, 1\}^*$ of length $2k \cdot \lfloor \log_2 |x| \rfloor$, where x is an instance of MAX-LIN2$_{\text{TLB}}$. Let the equations be numbered from 1 to m, then we can express a sequence of equations $e_1, \ldots e_l$, as a sequence of k integers between 0 and m (if $l < k$ then we end the sequence with $k - l$ zeroes). Each integer between 0 and m can be expressed by a string in $\{0, 1\}^*$ of length at most $\lceil \log_2 m \rceil \leq \lceil \log_2 |x| \rceil$, so certainly it can be expressed by a string of length $2 \lfloor \log_2 |x| \rfloor$. Therefore we can express the k integers as a string of length $2k \cdot \lfloor \log_2 |x| \rfloor$.

For an instance x of MAX-LIN2$_{\text{TLB}}$ and a string $y \in \{0, 1\}^*$, let us call y a *certificate for x* if $|y| = 2k \cdot \lfloor \log_2 |x| \rfloor$ and y encodes a sequence of k integers corresponding to a sequence of equations $e_1, \ldots, e_l$ in x, such that by marking each equation in turn in iterations of $\mathcal{H}$, we get a set of marked equations of weight at least k. It follows that x is a YES-instance if and only if there exists a certificate for x. Furthermore we can check in polynomial time whether y is a certificate of x by trying to convert y into a sequence of equations and running algorithm $\mathcal{H}$ marking those equations. (This is in fact a stronger result than we require for this proof - we only need that the algorithm is fixed-parameter tractable rather than polynomial.)

We now let

$$Y = \{(x, y) \in \Sigma^* \times \{0, 1\}^* | x \text{ is a YES-instance of MAX-LIN2}_{\text{TLB}} \text{ and } y \text{ is a certificate of } x\}$$

and let Q be the set of all YES-instances of MAX-LIN2$_{\text{TLB}}$. By definition of Y and the definition of a certificate, conditions (ii) and (iii) of Lemma 7.2.12 are satisfied. As we can determine in polynomial time whether y is a certificate for x, condition (i) is also satisfied. Therefore, by Lemma 7.2.12, MAX-LIN2$_{\text{TLB}}$ is in W[P].

□

Max-r-Lin2$_{\text{TLB}}$, Max-r-Sat$_{\text{TLB}}$ and Max r-CSP AA

Using Theorem 7.2.6 we can prove the following two results.

Corollary 7.2.13. *Let $r \geq 2$ be a fixed integer. Then* Max-r-Lin2$_{\text{TLB}}$ *has a kernel with $O(k \log k)$ variables and can be solved in time $2^{O(k \log k)} + m^{O(1)}$.*

Proof. Observe that $m \leq n^r$ and $n^r \leq 2^{n/(k-1)} - 2$ if $n \geq c(r)k \log_2 k$ provided $c(r)$ is large enough ($c(r)$ depends only on r). Thus, by Theorem 7.2.6, if $n \geq c(r)k \log_2 k$ then the answer to Max-r-Lin2$_{\text{TLB}}$ is yes. Hence, we obtain a problem kernel with at most $c(r)k \log_2 k = O(k \log k)$ variables and, therefore, can solve Max-r-Lin2$_{\text{TLB}}$ in time $2^{O(k \log k)} + m^{O(1)}$. □

Corollary 7.2.14. *Let $r \geq 2$ be a fixed integer. Then there is a bikernel from* Max-r-Sat$_{\text{TLB}}$ *to* Max-r-Lin2$_{\text{TLB}}$ *with $O(k \log k)$ variables. Moreover,* Max Exact r-SAT *has a kernel with $O(k \log k)$ variables and can be solved in time $2^{O(k \log k)} + m^{O(1)}$.*

Proof. Let F be an r-CNF formula with clauses $C_1, \dots, C_m$ in the variables $x_1, x_2, \dots, x_n$. We may assume that $x_i \in \{-1, 1\}$, where -1 corresponds to true. For F, following [5] consider

$$g(x) = \sum_{C \in F} [1 - \prod_{x_i \in \text{var}(C)} (1 + \varepsilon_i x_i)],$$

where var(C) is the set of variables of C, $\varepsilon_i \in \{-1, 1\}$ and $\varepsilon_i = 1$ if and only if x_i is in C. It is shown in [5] that the answer to Max Exact r-SAT is yes if and only if there is a truth assignment x^0 such that $g(x^0) \geq k$.

Algebraic simplification of $g(x)$ will lead us to Fourier expansion of $g(x)$:

$$g(x) = \sum_{S \in \mathcal{F}} c_S \prod_{i \in S} x_i, \tag{7.4}$$

where $\mathcal{F} = \{\emptyset \neq S \subseteq [n] : c_S \neq 0, |S| \leq r\}$. Thus, $|\mathcal{F}| \leq n^r$. By Proposition 7.2.7, $\sum_{S \in \mathcal{F}} c_S \prod_{i \in S} x_i$ can be viewed as an instance of Max r-Lin and, thus, we can reduce Max Exact r-SAT into Max r-Lin in polynomial time (the algebraic simplification can be done in polynomial time as r is fixed). By Corollary 7.2.13, Max r-Lin has a kernel with $O(k \log k)$ variables. This kernel is a bikernel from Max Exact r-SAT to Max r-Lin. Using this bikernel, we can solve Max Exact r-SAT in time $2^{O(k \log k)} + m^{O(1)}$.

It remains to use the transformation described in [5] of a bikernel from Max Exact r-SAT to Max r-Lin into a kernel of Max Exact r-SAT. This transformation gives us a kernel with $O(k \log k)$ variables. □

In the Boolean Max-r-Constraint Satisfaction Problem (Max-r-CSP), we are given a collection of Boolean functions, each involving at most r variables, and asked to find a truth assignment that satisfies as many functions as possible. We will consider the following parameterized version of Max-r-CSP. We are given a set Φ of Boolean functions, each involving at most r variables, and a collection $\mathcal{F}$ of m Boolean functions, each $f \in \mathcal{F}$ being a member of Φ, and each acting on some subset of the n Boolean variables $x_1, x_2, \ldots, x_n$ (each $x_i \in \{-1, 1\}$). We are to decide whether there is a truth assignment to the n variables such that the total number of satisfied functions is at least $E + k2^{-r}$, where E is the average value of the number of satisfied functions.

Corollary 7.2.15. *Let $r \geq 2$ be a fixed integer. Then there is a bikernel from* Max-r-Sat$_{\text{TLB}}$ *to* Max-r-Lin2$_{\text{TLB}}$ *with $O(k \log k)$ variables.* Max r-CSP *can be solved in time $2^{O(k \log k)} + m^{O(1)}$.*

Proof. Following [6] for a boolean function f of $r(f) \leq r$ boolean variables $x_{i_1}, \ldots, x_{i_{r(f)}}$, introduce a polynomial $h_f(x),\ x = (x_1, x_2, \ldots, x_n)$ as follows. Let $V_f \subset \{-1, 1\}^{r(f)}$ denote the set of all satisfying assignments of f. Then

$$h_f(x) = 2^{r-r(f)} \sum_{(v_1,\ldots,v_{r(f)}) \in V_f} [\prod_{j=1}^{r(f)} (1 + x_{i_j} v_j) - 1].$$

Let $h(x) = \sum_{f \in \mathcal{F}} h_f(x)$. It is easy to see (cf. [5]) that the value of $h(x)$ at x^0 is precisely $2^r(s - E)$, where s is the number of the functions satisfied by the truth assignment x^0, and E is the average value of the number of satisfied functions. Thus, the answer to Max-r-CSP is yes if and only if there is a truth assignment x^0 such that $h(x^0) \geq k$. The rest of the proof is similar to that of Corollary 7.2.14. □

Chapter 8

Future Work

In this thesis, we examined some digraph problems from parameterized perspective. We also considered a new type of parameterization for constraint satisfaction problems and investigated their parameterized complexity. As we wrap up the thesis, we propose some open problems for future work.

Parameterized Digraph Problems

The $O^*(3.72^k)$-algorithm in Chapter 3 is the current best for Directed k-Leaf and obviously it can be improved. For undirected graphs, k-Leaf allows $O^*(3.4575^k)$-algorithm by Raible and Fernau [103]. Their algorithm employs the new paradigm of Measure & Conquer to further develop the algorithm of [82]. Introducing non-standard measure for sophisticated analysis of search-tree based algorithm, known as Measure & Conquer, turned out to be a powerful approach. Although our algorithm for Directed k-Leaf uses non-standard measure implicitly, this possibility can be pursued more aggressively. Another interesting question is whether the quadratic kernel for Rooted Directed k-Leaf [42] can be improved to a linear one. It seems that the quadratic order is inevitable with respect to the reduction rules presented in [42]. We may need a new insight to answer this question.

For the problems k-Path and k-Out-Tree, there is a significant gap between the best randomized algorithm of running time $O^*(2^k)$ and deterministic algorithms of running time $O^*(4^k)$ and $O^*(6.14^k)$. The randomized algorithm [107, 84] converts k-Path (and other parameterized problems) into the problem of detecting a multilinear monomial in a degree-k polynomial, which is represented as a canonical arithmetic circuit. Then, a suitable algebraic structure provides elements to the variables so that a k-multilinear monomial is 'properly colored' with good probability by a random assignment of elements. To check whether some monomial is 'properly colored', polynomial identity testing is executed. The natural questions are whether such an algorithm can be made faster and whether it can be derandomized. The answer is negative for both questions if one considers the approach as is suggested in [107, 84]. Their randomized algorithm is equipped with polynomial identity testing, and polynomial-time derandomization of it implies strong circuit lower bounds [107]. Moreover, it is proved in [84] that essentially $O^*(2^k)$ cannot be improved by choosing a different algebraic structure. The early $O^*(2^{3k/2})$-algorithm of [83] looks more amenable to derandomization as it does not come with polynomial identity testing.

On the other hand, the deterministic $O^*(4^k)$-algorithm [31] and our $O^*(6.14^k)$-algorithm use divide-and-conquer strategy. In every step, the length of the path we want to find halves and we explore (roughly) 2^k possibilities that a k-path is shared by two disjoint parts of the input graph. As a result, it seems inevitable to have constant of 4 within the current frame of divide-and-conquer. For k-Out-Tree, the situation is similar except that the size of the (out-)tree we want to find does not halve. With this difference in mind, 6.14 is essentially the best possible constant as well. As is pointed out in [83], one possibility to improve the constant is to reduce the number 2^k of trials in every step, but it would require a complicated re-usage of the computation. Even if this is possible, which we doubt, the resulting running time would be $O^*((2 + \varepsilon)^k)$. To summarize, we believe that closing the gap between the best randomized and deterministic algorithms would require a novel idea. We also mention the recent breakthrough result by Björklund [19] for Hamiltonicity Detection. The author presents a Monte Carlo algorithm of running time $O^*(1.657^n)$. Inspired by this, obviously one can ask if 2^k barrier is truly impregnable for k-Path and k-Out-Tree.

The problem Directed k-Internal is much related to k-Out-Tree. The algorithm in Chapter 4 also exploits this observation. Moreover, the recent result by Fomin et al. [58] successfully extends the idea of divide-and-conquer developed for k-Path and k-Out-Tree [31, 36] and applies it to Directed k-Internal, which allows them to avoid exponentially many iterations of k-Out-Tree algorithm as we did in Section 5.6. It is quite likely that a new idea and consequent algorithmic improvement for k-Path and k-Out-Tree will give an initiative for a better algorithm on Directed k-Internal. On the other hand, aiming at a sub-quadratic kernel for Directed k-Internal is indeed a worthwhile challenge. This open problem is also motivated by the result of [56], which exhibited a $3k$-kernel of the problem k-Internal Spanning Tree for undirected graphs.

Parameterized Permutation CSPs Above Average

The problems Linear Ordering and Betweenness from Chapter 6 can be viewed as a part of wider family of *permutation constraint satisfaction problems*. Let $\mathcal{S}_r$ be the set of all permutations on $\{1, 2, \ldots, r\}$. A permutation constraint satisfaction problem (PERMCSP) of arity r is specified by a subset $\Pi \subseteq \mathcal{S}_r$. An instance of PERMCSP consists of a set V of n variables and a *constraint set over* V, which is a multiset of ordered r-tuples of V. The goal of PERMCSP(Π) is to find a linear ordering α of V that maximizes the number of Π-satisfied constraints. Here, a constraint C is Π-satisfied by α if C follows a permutation in Π under α.

Now LINEAR ORDERING and BETWEENNESS are the problem PERMCSP(Π), where Π are $\{12\}$ and $\{123, 321\}$ respectively. Observe that the probability of a constraint to be Π-satisfied by a random ordering α equals $|\Pi|/r!$. Hence, there is a linear ordering α satisfying $|\Pi|/r!$ fraction of the constraints. It is not known whether any polynomial-time approximation for PERMCSP(Π) beyond this obvious threshold is attainable, and it's conjectured to be approximation resistant, see [30]. In case of $r \leq 3$, it is known that approximating beyond the fraction $|\Pi|/r!$ is Unique-Games hard [66, 30].

For the special case $\Pi = \{12\}$ or $\{123, 321\}$, the parameterized problem to satisfy k additional constraints beyond the average is fixed-parameter tractable by the result of Chapter 6. The obvious next step is to extend the parameterized problem for general Π. In other words, given a constraint set C, we ask if $\frac{|\Pi|}{r!}|C| + k$ constraints can be Π-satisfied by some linear ordering and whether this problem is fixed-parameter tractable. For $r = 3$, this problem is shown to be fixed-parameter tractable, and even to admit quadratic kernel in [72]. Getting beyond $r = 3$ seems to be far from trivial extension of the result in [72].

Parameterized CSPs Above SDP-based Approximation

Another interesting direction for research is the parameterized constraint satisfaction problems above the approximation guarantee by semi-definite programming (SDP) algorithm. For many constraint satisfaction problems, SDP-based randomized rounding algorithm provides the best known approximation ratio. Let us consider MAX CUT as an exemplary problem. This problem can be approximated within .878 of the optimum in polynomial time by randomly rounding the optimal solution of its semi-definite program formulation [62]. Since the work of [62], a vast body of literature has explored the power of SDP-based approach for better approximation. It is worth noting that SDP-based rounding algorithm guarantees an approximation ratio with respect to the SDP optimum. That is, .878-approximation by Goemans and Williamson guarantees a cut of size at least $.878 \cdot SDP$ for MAX CUT, where SDP is the optimal solution of the semi-definite program for MAX CUT. As we can solve SDP up to optimality (or at least, as close to the optimum as necessary for our purpose) in polynomial time, it is reasonable to ask how much does it take to obtain a solution strictly larger than this approximation guarantee. More precisely, we want to find a cut over which at least $.878 \cdot SDP + k$ edges cross, if one exists, and to know whether this can be done in fpt-time. Observe that the same question can be asked for general constraint satisfaction problems. A weaker version of the suggested parameterization has been studied by Kim and Williams [79], where we take the size *OPT* of an optimal cut in replacement of SDP.

The .878 ratio is only tight when the maxcut is about 84.4% of all edges, and for any other percentage, it is known that better approximations are possible. A series of works has focused on calculating the precise tradeoff between the maximum cut value of a graph and the achievable approximation ratio, culminating in an explicit determination of the tradeoff for every possible fraction of the cut value, assuming Unique Game Conjecture:

Theorem 8.0.16 (O'Donnell and Wu [98])**.** *For $\frac{1}{2} \leq s \leq c \leq 1$, we call the pair (c, s) an SDP gap if there exists a graph G with the SDP optimal value at least c and the size of a maximum cut at most s. The SDP gap curve is defined by $Gap_{SDP}(c) = \inf\{s : (c, s)$ is an SDP gap$\}$. Then there is a function $S : [\frac{1}{2}, 1] \to [\frac{1}{2}, 1]$ such that $Gap_{SDP}(c) = S(c)$ for all c. Here, c and s respectively denote the fraction of SDP optimal value and maximum cut in the sum of edge weights in G.*

Let $S : [\frac{1}{2}, 1] \to [\frac{1}{2}, 1]$ be any function such that when we are given an m-edge graph with an optimal SDP value of cm, it is possible to efficiently find a cut of size $S(c)m$ using some SDP rounding. Can we find a cut of size $S(c)m + k$, provided it exists? We believe that this problem should be fixed-parameter tractable.

Bibliography

[1] N. Alon, M. Bădoiu, E. D. Demaine, M. Farach-Colton, M. Hajiaghayi, and A. Sidiropoulos. Ordinal embeddings of minimum relaxation: general properties, trees, and ultrametrics. *ACM Trans. Algorithms*, 4(4):Art. 46, 21, 2008.

[2] N. Alon, F. V. Fomin, G. Gutin, M. Krivelevich, and S. Saurabh. Better algorithms and bounds for directed maximum leaf problems. In *FSTTCS 2007: Foundations of software technology and theoretical computer science*, volume 4855 of *Lecture Notes in Comput. Sci.*, pages 316–327. Springer, Berlin, 2007.

[3] N. Alon, F. V. Fomin, G. Gutin, M. Krivelevich, and S. Saurabh. Parameterized algorithms for directed maximum leaf problems. In *ICALP*, volume 4596 of *Lecture Notes in Comput. Sci.*, pages 352–362. Springer, Berlin, 2007.

[4] N. Alon, F. V. Fomin, G. Gutin, M. Krivelevich, and S. Saurabh. Spanning directed trees with many leaves. *SIAM J. Discrete Math.*, 23(1):466–476, 2008/09.

[5] N. Alon, G. Gutin, E. J. Kim, S. Szeider, and A. Yeo. Solving MAX-r-SAT above a tight lower bound. In *Algorithmica*, volume To appear, 2010.

[6] N. Alon, G. Gutin, and M. Krivelevich. Algorithms with large domination ratio. *J. Algorithms*, 50(1):118–131, 2004.

[7] N. Alon and J. H. Spencer. *The probabilistic method.* Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons Inc., Hoboken, NJ, third edition, 2008. With an appendix on the life and work of Paul Erdős.

[8] N. Alon, R. Yuster, and U. Zwick. Color-coding. *J. Assoc. Comput. Mach.*, 42(4):844–856, 1995.

[9] H. Alt, N. Blum, K. Mehlhorn, and M. Paul. Computing a maximum cardinality matching in a bipartite graph in time $O(n^{1.5}\sqrt{m/\log n})$. *Inform. Process. Lett.*, 37(4):237–240, 1991.

[10] B. Aspvall, M. F. Plass, and R. E. Tarjan. A linear-time algorithm for testing the truth of certain quantified Boolean formulas. *Inform. Process. Lett.*, 8(3):121–123, 1979.

[11] A. S. Asratian, T. M. J. Denley, and R. Häggkvist. *Bipartite graphs and their applications*, volume 131 of *Cambridge Tracts in Mathematics*. Cambridge University Press, Cambridge, 1998.

[12] M. Bădoiu, E. D. Demaine, M. Hajiaghayi, A. Sidiropoulos, and M. Zadimoghaddam. Ordinal embedding: Approximation algorithms and dimensionality reduction. In *APPROX-RANDOM*, pages 21–34, 2008.

[13] J. Bang-Jensen and G. Gutin. *Digraphs*. Springer Monographs in Mathematics. Springer-Verlag London Ltd., London, second edition, 2009. Theory, algorithms and applications.

[14] J. Barát. Directed path-width and monotonicity in digraph searching. *Graphs Combin.*, 22(2):161–172, 2006.

[15] C. Berge. Sur le couplage maximum d'un graphe. *C. R. Acad. Sci. Paris*, 247:258–259, 1958.

[16] D. Berwanger, A. Dawar, P. Hunter, and S. Kreutzer. DAG-width and parity games. In *STACS 2006*, volume 3884 of *Lecture Notes in Comput. Sci.*, pages 524–536. Springer, Berlin, 2006.

[17] T. Beyer and S. M. Hedetniemi. Constant time generation of rooted trees. *SIAM J. Comput.*, 9(4):706–712, 1980.

[18] Y. Bilu and N. Linial. Monotone maps, sphericity and bounded second eigenvalue. *J. Combin. Theory Ser. B*, 95(2):283–299, 2005.

[19] A. Björklund. Determinant sums for undirected hamiltonicity. In *FOCS*, page To appear, 2010.

[20] J. Blum, M. Ding, A. Thaeler, and X. Cheng. Connected dominating set in sensor networks and MANETs. In *Handbook of combinatorial optimization. Supplement Vol. B*, pages 329–369. Springer, New York, 2005.

[21] T. S. Blyth and E. F. Robertson. *Basic linear algebra*. Springer Undergraduate Mathematics Series. Springer-Verlag London Ltd., London, 1998. Revised reprint of ıt Matrices and vector spaces [Chapman and Hall, New York, 1986; MR0866312 (89b:00001b)].

[22] H. L. Bodlaender. Treewidth: Algorithmic techniques and results. pages 19–36, 1997.

[23] H. L. Bodlaender, R. G. Downey, M. R. Fellows, and D. Hermelin. On problems without polynomial kernels (extended abstract). In *ICALP (1)*, pages 563–574, 2008.

[24] H. L. Bodlaender, S. Thomassé, and A. Yeo. Kernel bounds for disjoint cycles and disjoint paths. In *ESA*, pages 635–646, 2009.

[25] A. Bonami. Étude des coefficients de Fourier des fonctions de $L^p(G)$. *Ann. Inst. Fourier*, 20(2):335–402, 1970.

[26] J. A. Bondy and U. S. R. Murty. *Graph theory*, volume 244 of *Graduate Texts in Mathematics*. Springer, New York, 2008.

[27] P. S. Bonsma, T. Brueggemann, and G. J. Woeginger. A faster FPT algorithm for finding spanning trees with many leaves. In *Mathematical foundations of computer science 2003*, volume 2747 of *Lecture Notes in Comput. Sci.*, pages 259–268. Springer, Berlin, 2003.

[28] P. S. Bonsma and F. Dorn. An fpt algorithm for directed spanning k-leaf. *CoRR*, abs/0711.4052, 2007.

[29] P. S. Bonsma and F. Dorn. Tight bounds and a fast fpt algorithm for directed max-leaf spanning tree. In *ESA*, pages 222–233, 2008.

[30] M. Charikar, V. Guruswami, and R. Manokaran. Every permutation CSP of arity 3 is approximation resistant. In *Computational Complexity, 2009. CCC '09. 24th Annual IEEE Conference on*, pages 62–73, July 2009.

[31] J. Chen, J. Kneis, S. Lu, D. Mölle, S. Richter, P. Rossmanith, S.-H. Sze, and F. Zhang. Randomized divide-and-conquer: improved path, matching, and packing algorithms. *SIAM J. Comput.*, 38(6):2526–2547, 2009.

[32] B. Chor, M. Fellows, and D. Juedes. Linear kernels in linear time, or how to save k colors in $O(n^2)$ steps. In *Graph-theoretic concepts in computer science*, volume 3353 of *Lecture Notes in Comput. Sci.*, pages 257–269. Springer, Berlin, 2004.

[33] B. Chor and M. Sudan. A geometric approach to betweenness. *SIAM J. Discrete Math.*, 11(4):511–523 (electronic), 1998.

[34] F. R. K. Chung. Separator theorems and their applications. In *Paths, flows, and VLSI-layout (Bonn, 1988)*, volume 9 of *Algorithms Combin.*, pages 17–34. Springer, Berlin, 1990.

[35] D. A. Cohen, M. C. Cooper, P. G. Jeavons, and A. A. Krokhin. The complexity of soft constraint satisfaction. *Artificial Intelligence*, 170(11):983–1016, 2006.

[36] N. Cohen, F. V. Fomin, G. Gutin, E. J. Kim, S. Saurabh, and A. Yeo. Algorithm for finding k-vertex out-trees and its application to k-internal out-branching problem. *J. Comput. Syst. Sci.*, 76(7):650–662, 2010.

[37] D. Coppersmith. Solving linear equations over gf(2): block lanczos algorithm. *Linear Algebra and its Applications*, 192:33 – 60, 1993.

[38] D. Cox, M. Burmeister, E. Price, S. Kim, and R. Myers. Radiation hybrid mapping: a somatic cell genetic method for constructing high-resolution maps of mammalian chromosomes. *Science*, 250(4978):245–250, 1990.

[39] R. Crowston, G. Gutin, and M. Jones. Note on max lin-2 above average. *Inf. Process. Lett.*, 110(11):451–454, 2010.

[40] R. Crowston, G. Gutin, M. Jones, E. J. Kim, and I. Z. Ruzsa. Systems of linear equations over $\mathbb{F}_2$ and problems parameterized above average. In *SWAT*, pages 164–175, 2010.

[41] J. Daligault, G. Gutin, E. J. Kim, and A. Yeo. Fpt algorithms and kernels for the directed k-leaf problem. *J. Comput. Syst. Sci.*, 76(2):144–152, 2010.

[42] J. Daligault and S. Thomassé. On finding directed trees with many leaves. In *IWPEC*, pages 86–97, 2009.

[43] P. Dankelmann, G. Gutin, and E. J. Kim. On complexity of minimum leaf out-branching problem. *Discrete Appl. Math.*, 157(13):3000–3004, 2009.

[44] R. de Wolf. A brief introduction to fourier analysis on the boolean cube. *Theory of Computing, Graduate Surveys*, 1:1–20, 2008.

[45] A. Demers and A. Downing. Minimum leaf spanning tree. In *US Patent*, volume 6,105,018. 2000.

[46] R. Diestel. *Graph theory*, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag, Berlin, third edition, 2005.

[47] G. Ding, T. Johnson, and P. Seymour. Spanning trees with many leaves. *J. Graph Theory*, 37(4):189–197, 2001.

[48] R. Downey and M. Fellows. *Parameterized Complexity*. Springer-Verlag, New York, 1999.

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

[50] S. E. Dreyfus and R. A. Wagner. The Steiner problem in graphs. *Networks*, 1:195–207, 1971/72.

[51] V. Estivill-Castro, M. R. Fellows, M. A. Langston, and F. A. Rosamond. Fpt is p-time extremal structure i. In *ACiD*, pages 1–41, 2005.

[52] M. Fellows, P. Heggernes, F. Rosamond, C. Sloper, and J. A. Telle. Finding k disjoint triangles in an arbitrary graph. In *Graph-theoretic concepts in computer science*, volume 3353 of *Lecture Notes in Comput. Sci.*, pages 235–244. Springer, Berlin, 2004.

[53] M. R. Fellows, C. McCartin, F. A. Rosamond, and U. Stege. Coordinatized kernels and catalytic reductions: an improved FPT algorithm for max leaf spanning tree and other problems. In *FST TCS 2000: Foundations of software technology and theoretical computer science (New Delhi)*, volume 1974 of *Lecture Notes in Comput. Sci.*, pages 240–251. Springer, Berlin, 2000.

[54] H. Fernau, F. V. Fomin, D. Lokshtanov, D. Raible, S. Saurabh, and Y. Villanger. Kernel(s) for problems with no kernel: On out-trees with many leaves. In *STACS*, pages 421–432, 2009.

[55] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer-Verlag, 2006.

[56] F. V. Fomin, S. Gaspers, S. Saurabh, and S. Thomassé. A linear vertex kernel for maximum internal spanning tree. In *ISAAC*, pages 275–282, 2009.

[57] F. V. Fomin, F. Grandoni, and D. Kratsch. Solving connected dominating set faster than 2^n. *Algorithmica*, 52(2):153–166, 2008.

[58] F. V. Fomin, D. Lokshtanov, F. Grandoni, and S. Saurabh. Sharp separation and applications to exact and parameterized algorithms. In *LATIN*, pages 72–83, 2010.

[59] M. L. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with 0(1) worst case access time. *J. ACM*, 31(3):538–544, 1984.

[60] G. Galbiati, A. Morzenti, and F. Maffioli. On the approximability of some maximum spanning tree problems. *Theoret. Comput. Sci.*, 181(1):107–118, 1997. Latin American Theoretical INformatics (Valparaíso, 1995).

[61] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoret. Comput. Sci.*, 1(3):237–267, 1976.

[62] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145, 1995.

[63] S. Goss and H. Harris. New methods for mapping genes in human chromosomes. *Nature*, 255:680–684, 1975.

[64] L. Gross. Logarithmic Sobolev inequalities. *Amer. J. Math.*, 97:1061–1083, 1975.

[65] J. Guo and R. Niedermeier. Invitation to data reduction and problem kernelization. *SIGACT News*, 38(1):31–45, 2007.

[66] V. Guruswami, R. Manokaran, and P. Raghavendra. Beating the random ordering is hard: Inapproximability of maximum acyclic subgraph. In *FOCS*, pages 573–582, 2008.

[67] G. Gutin, E. J. Kim, M. Mnich, and A. Yeo. Betweenness parameterized above tight lower bound. *J. Comput. Syst. Sci.*, To appear, 2010.

[68] G. Gutin, E. J. Kim, S. Szeider, and A. Yeo. A probabilistic approach to problems parameterized above or below tight bounds. *J. Comput. Syst. Sci.*, To appear, 2010.

[69] G. Gutin, A. Rafiey, S. Szeider, and A. Yeo. The linear arrangement problem parameterized above guaranteed value. *Theory Comput. Syst.*, 41(3):521–538, 2007.

[70] G. Gutin, I. Razgon, and E. J. Kim. Minimum leaf out-branching and related problems. *Theoret. Comput. Sci.*, 410(45):4571–4579, 2009.

[71] G. Gutin, S. Szeider, and A. Yeo. Fixed-parameter complexity of minimum profile problems. *Algorithmica*, 52(2):133–152, 2008.

[72] G. Gutin, L. van Iersel, M. Mnich, and A. Yeo. All ternary permutation constraint satisfaction problems parameterized above average have polynomial kernels. Tech. Report at http://arxiv.org/abs/1004.1956, 2010.

[73] J. Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, 2001.

[74] J. Håstad and S. Venkatesh. On the advantage over a random assignment. *Random Structures Algorithms*, 25(2):117–149, 2004.

[75] K. Iwama. Cnf satisfiability test by counting and polynomial average time. *SIAM J. Comput.*, 18(2):385–391, 1989.

[76] T. Johnson, N. Robertson, P. D. Seymour, and R. Thomas. Directed tree-width. *J. Comb. Theory, Ser. B*, 82(1):138–154, 2001.

[77] S. Jukna. *Extremal combinatorics*. Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 2001. With applications in computer science.

[78] M. Karpinski and W. Schudy. Approximation schemes for the betweenness problem in tournaments and related ranking problems. Tech. Report at http://arxiv.org/abs/0911.2214, 2009.

[79] E. J. Kim and R. Williams. Improved parameterized algorithms for constraint satisfaction. Tech. Report at http://arxiv.org/abs/1008.0213, 2010.

[80] D. J. Kleitman, J. Shearer, and D. Sturtevant. Intersections of k-element sets. *Combinatorica*, 1(4):381–384, 1981.

[81] D. J. Kleitman and D. B. West. Spanning trees with many leaves. *SIAM J. Discrete Math.*, 4(1):99–106, 1991.

[82] J. Kneis, A. Langer, and P. Rossmanith. A new algorithm for finding trees with many leaves. In *ISAAC*, pages 270–281, 2008.

[83] I. Koutis. Faster algebraic algorithms for path and packing problems. In *ICALP (1)*, pages 575–586, 2008.

[84] I. Koutis and R. Williams. Limits and applications of group algebras for parameterized problems. In *ICALP (1)*, pages 653–664, 2009.

[85] S. Kreutzer and S. Ordyniak. Digraph decompositions and monotonicity in digraph searching. In *WG*, pages 336–347, 2008.

[86] A. Krokhin, P. Jeavons, and P. Jonsson. Constraint satisfaction problems on intervals and lengths. *SIAM J. Discrete Math.*, 17(3):453–477 (electronic), 2004.

[87] M. Lampis, G. Kaouri, and V. Mitsou. On the algorithmic effectiveness of digraph decompositions and complexity measures. In *ISAAC*, pages 220–231, 2008.

[88] N. Linial and D. Sturtevant. Unpublished result. 1987.

[89] H.-I. Lu and R. Ravi. Approximating maximum leaf spanning trees in almost linear time. *J. Algorithms*, 29(1):132–141, 1998.

[90] M. Mahajan and V. Raman. Parameterizing above guaranteed values: MaxSat and MaxCut. *J. Algorithms*, 31(2):335–354, 1999.

[91] M. Mahajan, V. Raman, and S. Sikdar. Parameterizing above or below guaranteed values. *J. Comput. System Sci.*, 75(2):137–153, 2009.

[92] Y. Makarychev. Simple linear time approximation algorithm for betweenness. Technical Report MSR-TR-2009-74, Microsoft Research New England, June 2009.

[93] M. Naor, L. J. Schulman, and A. Srinivasan. Splitters and near-optimal derandomization. In *FOCS*, pages 182–191, 1995.

[94] R. Niedermeier. *Invitation to fixed-parameter algorithms*, volume 31 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, Oxford, 2006.

[95] A. Nilli. Perfect hashing and probability. *Combinatorics, Probability & Computing*, 3:407–409, 1994.

[96] J. Obdrzálek. Dag-width: connectivity measure for directed graphs. In *SODA*, pages 814–821, 2006.

[97] R. O'Donnell. Some topics in analysis of boolean functions. In *STOC*, pages 569–578, 2008.

[98] R. O'Donnell and Y. Wu. An optimal sdp algorithm for max-cut, and equally optimal long code tests. In *STOC*, pages 335–344, 2008.

[99] J. Opatrný. Total ordering problem. *SIAM J. Comput.*, 8(1):111–114, 1979.

[100] R. Otter. The number of trees. *Ann. Math.*, 49:583–599, 1948.

[101] E. Prieto and C. Sloper. Either/or: Using vertex cover structure in designing fpt-algorithms - the case of k-internal spanning tree. In *WADS*, pages 474–483, 2003.

[102] E. Prieto and C. Sloper. Reducing to independent set structure – the case of k-internal spanning tree. *Nord. J. Comput.*, 12(3):308–318, 2005.

[103] D. Raible and H. Fernau. An amortized search tree analysis for *k*-leaf spanning tree. In *SOFSEM*, pages 672–684, 2010.

[104] R. Solis-Oba. 2-approximation algorithm for finding a spanning tree with maximum number of leaves. In *ESA*, pages 441–452, 1998.

[105] Y. Villanger, P. Heggernes, C. Paul, and J. A. Telle. Interval completion is fixed parameter tractable. *SIAM J. Comput.*, 38(5):2007–2020, 2008/09.

[106] V. Vovk. Private communication. August, 2009.

[107] R. Williams. Finding paths of length k in $O^*(2^k)$ time. *Inf. Process. Lett.*, 109(6):315–318, 2009.