New Methods in Parameterized Algorithms and Complexity

Daniel Lokshtanov

Dissertation for the degree of Philosophiae Doctor (PhD)

University of Bergen
Norway

April 2009

Acknowledgements

First of all, I would like to thank my supervisor Pinar Heggernes for her help and guidance, both with my reasearch and with all other aspects of being a graduate student. This thesis would not have existed without her.

Already in advance, I would like to thank my PhD committee, Maria Chudnovsky, Rolf Niedermeier and Sverre Storøy.

I also want to thank all my co-authors, Noga Alon, Oren Ben-Zwi, Hans Bodlaender, Michael Dom, Michael Fellows, Henning Fernau, Fedor V. Fomin, Petr A. Golovach, Fabrizio Grandoni, Pinar Heggernes, Danny Hermelin, Jan Kratochvíl, Elena Losievskaja, Federico Mancini, Rodica Mihai, Neeldhara Misra, Matthias Mnich, Ilan Newman, Charis Papadopoulos, Eelko Penninkx, Daniel Raible, Venkatesh Raman, Frances Rosamond, Saket Saurabh, Somnath Sikdar, Christian Sloper, Stefan Szeider, Jan Arne Telle, Dimitrios Thilikos, Carsten Thomassen and Yngve Villanger. A special thanks goes to Saket Saurabh, with whom I have shared many sleepless nights and cups of coffee preparing this work.

Thanks to my family, especially to my mother and father. In more than one way, i am here largely because of you.

I would also like to thank my friends for filling my life with things other than discrete mathematics. You mean a lot to me, hugs to all of you and kisses to one in particular.

Thanks also to the Algorithms group at the University of Bergen, you are a great lot. An extra thanks goes to my table-tennis opponents, even though this thesis exists despite of your efforts to keep me away from it.

Finally, thanks to Pasha for being furry and having whiskers.

Bergen, April 2009
Daniel Lokshtanov

Contents

Chapter 1

Introduction

Parameterized Algorithms and Complexity is a natural way to cope with problems that are considered intractable according to the classical P versus NP dichotomy. In practice, large instances of NP-complete problems are solved exactly every day. The reason for why this is possible is that problem instances that occur naturally often have a hidden structure. The basic idea of Parameterized Algorithms and Complexity is to extract and harness the power of this structure. In Parameterized Algorithms and Complexity every problem instance comes with a relevant secondary measurement k, called a *parameter*. The parameter measures how difficult the instance is, the larger the parameter, the harder the instance. The hope is that at least for small values of the parameter, the problem instances can be solved efficiently. This naturally leads to the definition of *fixed parameter tractability*. We say that a problem is fixed parameter tractable (FPT) if problem instances of size n can be solved in $f(k)n^{O(1)}$ time for some function f independent of n.

Over the last two decades, Parameterized Algorithms and Complexity has established itself as an important subfield of Theoretical Computer Science. Parameterized Algorithms has its roots in the Graph Minors project of Robertson and Seymour [122], but in fact, such algorithms have been made since the early seventees [54, 108]. In the late eightees several FPT algorithms were discovered [61, 59] but despite considerable efforts no fixed parameter tractable algorithm was found for problems like Independent Set and Dominating Set. In the early ninetees, Downey and Fellows [51, 50] developed a framework for showing infeasibility of FPT algorithms for parameterized problems under certain complexity theoretic assumptions, and showed that the existence of FPT algorithms for Independent Set and Dominating Set is unlikely.

The possibility of showing lower as well as upper bounds sparked interest in the field, and considerable progress has been made in developing methods to show tractability and intractability of parameterized problems. Notable methods that have been developed include Bounded Search Trees, Kernelization, Color Coding and Iterative Compression. Over the last few years Kernelization has grown into a subfield of its own, due to a connection to polynomial time preprocessing and the techniques that have been developed over the last year for showing kernelization lower bounds [70, 20].

In this thesis we develop new methods for showing upper and lower bounds, both for Parameterized Algorithms and for Kernelization.

1.1 Overview of the Thesis

In Section 1.2 we give the necessary notation and definitions for the thesis. In Part I we give an overview of existing techniques in Parameterized Algorithms and Complexity. This part of the thesis is organized as follows. First we cover techniques for algorithm design, then we review existing medthods for showing running time lower bounds up to certain complexity-theoretic assumptions. Then we turn our attention to *kernelization* or polynomial time preprocessing. In Chapter 4 we describe the most well-known techniques for proving kernelization results. In Chapter 5 we survey the recent methods developed for giving kernelization lower bounds.

In Part II we describe new methods in Parameterized Algorithms and Complexity that were developed as part of the author's PhD research. The part is organized similarly to Part I, that is, we first consider algorithmic upper and lower bounds, before turning our attention to upper and lower bounds for kernelization. In Chapter 6 we describe a new color-coding based scheme to give subexponential time algorithms for problems on dense structures. In Chapter 7 we illustrate how explicit identification makes it easier to construct hardness reductions, and use this kind of reductions to show a trade-off between expressive power and computational tractability. In Chapter 8 we give two results. Our first result is a meta-theorem for kernelization of graph problems restricted to graphs embeddable into surfaces with constant genus. Our second result is a demonstration that a relaxed notion of kernelization called *Turing kernelization* indeed is more powerful than kernelization in the traditional sense. Finally, in Chapter 9 we show that explicit identification is useful not only in hardness reductions, but also to show kernelization lower bounds. Several such lower bounds are derived by applying the proposed method.

Parts of this thesis have been published as conference articles and some parts are excerpts from articles currently under peer review. Below we give an overview of the articles that were used as a basis for this work.

- Section 2.3 is based on the manuscript *Simpler Parameterized Algorithm for OCT*, coauthored with S. Saurabh and S. Sikdar.

- Section 2.8 is based on an excerpt from the article *Graph Layout problems*

Parameterized by Vertex Cover, coauthored with M. Fellows, N. Misra, F. A. Rosamond and S. Saurabh. Proceedings of ISAAC 2008, pages 294-305.

- Section 4.2 is based on an excerpt from the manuscript *Hypergraph Coloring meets Hypergraph Spanning Tree*, coauthored with S. Saurabh.
- Chapter 6 is based on the manuscript *Fast FAST*, coauthored with N. Alon and S. Saurabh.
- Section 7.1.1 is based on an excerpt from the article *On the Complexity of Some Colorful Problems Parameterized by Treewidth*, coauthored with M. Fellows, F. V. Fomin, F. Rosamond, S. Saurabh, S. Szeider and C. Thomassen. In the proceedings of COCOA 2007, pages 366-377. (Invited Paper.)
- Section 7.1.2 is based on an excerpt from the article *Capacitated Domination and Covering: A Parameterized Perspective*, coauthored with M. Dom, Y. Villanger and S. Saurabh. Proceedings of IWPEC 2008, pages 78-90.
- Section 7.2 is based on the article *Clique-width: On the Price of Generality*, coauthored with F. V. Fomin, P. A. Golovach and S.Saurabh. Proceedings of SODA 2009, pages 825-834.
- Section 8.1 is based on the manuscript *(Meta)-Kernelization*, coauthored with H. Bodlaender, F. V. Fomin, E. Penninkx, S. Saurabh and D. Thilikos.
- Sections 8.2 and 9.1 are based on the article *Kernel(s) for Problems With no Kernel: On Out-Trees With Many Leaves*, coauthored with H. Fernau, F. V. Fomin, D. Raible, S. Saurabh and Y. Villanger. Proceedings of STACS 2009, pages 421-432.
- Section 9.2 is based om the manuscript *Incompressibility through Colors and IDs*, coauthored with M. Dom and S. Saurabh.

1.2 Notation and Definitions

Basic Notions We assume that the reader is familiar with basic notions like sets, functions, polynomials, relations, integers etc. In particular, for these notions we follow the setup of [116].

Problems A *problem* or *language* is a set L of strings over a finite alphabet Σ, that is $L \subseteq (\Sigma)^*$. A string $s \in L$ is a *yes* instance of L and a string $s \notin L$ is a *no* instance of L. A *parameterized problem* is a subset Π of $(\Sigma)^* \times \mathbb{N}$, that is, every instance of Π comes with a natural number called the *parameter*. In an *optimization problem* we are given as input an instance s, defining a set $F(s)$ of *feasible solutions* for a problem specific function F. The objective is to find an element $x \in F(s)$ that minimizes or maximizes a certain problem spesific *objective function* $g(x)$. If the objective is to minimize (maximize) $g(x)$ the problem is referred to as a minimization (maximization) problem. The minimum (maximum) value of the objective function over all the feasible solutions for an instance s is called the *optimal value* of the objective function.

Function Growth We employ big-Oh notation which suppresses constant factors and lower order terms. That is, for two functions $f : \mathbb{N} \to \mathbb{N}$ and $g : \mathbb{N} \to \mathbb{N}$ we say $f(n) = O(g(n))$ if there are constants c_0, $c_1 > 0$ such that for all $x \geq c_0$, $f(x) \leq c_1 g(x)$. We say that a function f is *singly exponential* if $f = O(2^{n^c})$ for some constant c. If $f = O(2^{2^{n^c}})$ for some constant c we say f is *doubly exponential.* We will sometimes misuse notation and say that $f(n)$ is at least $O(g(n))$, meaning that $g(n) = O(f(n))$.

Algorithms An *algorithm for the problem L* is an algorithm that for a string s determines whether $s \in L$. The running time of the algorithm is measured in the number of steps the algorithm performs. We will assume a single processor, random-access machine as the underlying machine model throughout this thesis. In the random-access machine any simple operation (arithmetic, if-statements, memory-access etc.) takes unit length of time, and the word size is sufficiently large to hold numbers that are singly exponential in the size of the input. A *c-approximation algorithm* for a minimization problem is an algorithm that for a given instance finds a feasible solution x such that the value of the of the objective function $g(x)$ is at most c times the optimal value. Similarly a c-approximation algorithm for a maximization problem is an algorithm that for a given instance finds a feasible solution x such that the value of the of the objective function $g(x)$ is at least $1/c$ times the optimal value. An optimization problem that has a c-approximation algorithm is said to be c-approximable. A polynomial time approximation scheme (PTAS) for an optimization problem is an algorithm that for any fixed $\epsilon > 0$ computes a polynomial time $(1 + \epsilon)$-approximation algorithm for the problem.

Graphs A *graph* G is a set $V(G)$ of vertices and a set $E(G)$ of unordered pairs of vertices called *edges.* A vertex u and a vertex v are *adjacent* if the edge $uv \in E(G)$, and the vertices u and v are *incident* to the edge uv. The vertices u and v are referred to as the *endpoints* of the edge uv. If $vv \notin E(G)$ for all $v \in V(G)$ the graph G is called *simple.* Unless specifically stated otherwise the graphs considered in this thesis are simple. The *open neighbourhood* or just *neighbourhood* of a vertex v in the graph G is the set $N_G(v) = \{u\ :\ uv \in E(G)\}$

and the *closed neighbourhood* of a vertex v is the set $N_G[v] = N_G(v) \cup \{v\}$. The neighbourhood of a vertex set S is $N_G(S) = (\bigcup_{v \in S} N(v)) \setminus S$. For a vertex set $S \subset V(G)$, we define $\partial_G(S)$ as the set of vertices in S that have a neighbor in $V \setminus S$. The *degree* of a vertex v is $|N_G(v)|$. For $v \in V(G)$, by $E_G(v)$ we mean the set of edges incident to v. When the graph is clear from the context we will omit the subscripts.

A *walk* in a graph G is a sequence $v_1, \dots, v_k$ of vertices such that $v_i v_{i+1} \in E(G)$. A walk that never contains the same vertex twice is called a *path*. A walk where the first and the last vertex is the same but the other vertices are unique is called a *cycle*. The length of a walk is $k-1$. A *subgraph* of G is a graph G' such that $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$. The *subgraph induced* by a vertex set S is $G[S] = (S, \; \{uv \in E(G) \;:\; u \in S \wedge v \in S\})$. For an edge set S, by $G \setminus S$ we denote $G' = (V(G), E(G) \setminus S)$ and for a vertex set S', $G \setminus S$ we denote $G[V(G) \setminus S]$. For a vertex x and edge e, $G \setminus x$ and $G \setminus e$ is defined as $G \setminus \{x\}$ and $G \setminus \{e\}$ respectively. *Contracting* an edge uv of G amounts to removing the vertices u and v from the graph and adding a new vertex u' and making u' adjacent to $N(u) \cup N(v) \setminus \{u, v\}$. If H can be obtained from G by repeatedly contracting edges, we say that H is a *contraction* of G. If H is a subgraph of a contraction of G we say that H is a *minor* of G, and that $H \leq_M G$. A graph G is *connected* if there is a path between every pair of vertices. We will say that a vertex or edge set with a specific property is *minimal* (*maximal*) if no proper subset (superset) of the set has the property. A *connected component* of G is an induced subgraph C of G such that $V(C)$ is a maximal subset of $V(G)$ such that $G[V(C)]$ is connected. A set $S \subseteq V(G)$ is a *separator* if $G \setminus S$ is disconnected.

A set $S \subseteq V(G)$ of pairwise adjacent vertices is called a *clique* and a set of a set $S \subseteq V(G)$ of pairwise non-adjacent vertices is called an *independent set*. A set $S \subseteq V(G)$ such that every edge has an endpoint in S is called a vertex cover of G. A set S such that every vertex in $V(G) \setminus S$ has a neighbour in V is called a *dominating set* of G.

Treewidth A *tree decomposition* of a graph G is a pair (X, T) where T is a tree whose vertices we will call *nodes* and $X = (\{X_i \mid i \in V(T)\})$ is a collection of subsets of $V(G)$ such that

1. $\bigcup_{i \in V(T)} X_i = V(G)$,
2. for each edge $vw \in E(G)$, 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 $\{i \mid v \in X_i\}$ forms a subtree of T.

The *width* of a tree decomposition $(\{X_i \mid i \in V(T)\}, T)$ 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 notation $\mathbf{tw}(G)$ to denote the treewidth of a graph G. A tree decomposition is called a *nice tree decomposition* if the following conditions are satisfied: Every node of the tree T has at most two children; if a node t has two children t_1 and t_2, then $X_t = X_{t_1} = X_{t_2}$; and if a node t has one child t_1, then either $|X_t| = |X_{t_1}| + 1$ and $X_{t_1} \subset X_t$ or $|X_t| = |X_{t_1}| - 1$ and $X_t \subset X_{t_1}$. It is possible to transform a given tree decomposition into a nice tree decomposition in time $O(|V| + |E|)$ [18].

Clique-width Let G be a graph, and k be a positive integer. A k-graph is a graph whose vertices are labeled by integers from $\{1, 2, \ldots, k\}$. We call the k-graph consisting of exactly one vertex labeled by some integer from $\{1, 2, \ldots, k\}$ an initial k-graph. The *cliquewidth* $\mathbf{cwd}(G)$ is the smallest integer k such that a k-graph isomorphic to G can be constructed by means of repeated application of the following four operations:

1. *Introduce*, construction of an initial k-graph labeled by i, denoted by $i(v)$.
2. *Disjoint union*, denoted by $\oplus$.
3. *Relabel*, changing all labels i to j, denoted by $\rho_{i \to j}$.
4. *Join*, connecting all vertices labeled by i with all vertices labeled by j by edges, denoted by $\eta_{i,j}$.

An *e*xpression tree of a graph G is a rooted tree T of the following form:

- The nodes of T are of four types i, $\oplus$, η and ρ.
- Introduce nodes $i(v)$ are leaves of T, corresponding to initial k-graphs with vertices v, which are labeled i.
- A union node $\oplus$ stands for a disjoint union of graphs associated with its children.
- A relabel node $\rho_{i \to j}$ has one child and is associated with the k-graph, which is the result of relabeling operation for the graph corresponding to the child.
- A join node $\eta_{i,j}$ has one child and is associated with the k-graph, which is the result of join operation for the graph corresponding to the child.
- The graph G is isomorphic to the graph associated with the root of T (with all labels removed).

The *w*idth of the tree T is the number of different labels appearing in T. If a graph G has $\mathbf{cwd}(G) \leq k$ then it is possible to construct a rooted expression tree T with *width* k of G. A well-known fact is that if the treewidth of a graph is bounded then its cliquewidth also is bounded. On the other hand, complete graphs have clique-width 2 and unbounded treewidth.

Theorem 1.2.1 ([34]) *If graph G has treewidth at most t, then* $\mathbf{cwd}(G)$ *is at most $k = 3 \cdot 2^{t-1}$. Moreover, an expression tree for G of width at most k can be constructed in FPT time (with treewidth being the parameter) from the tree decomposition of G.*

The second claim in Theorem 1.2.1 is not given explicitly in [34]. However it can be shown since the upper bound proof in [34] is constructive (see also [37, 57]). Note that if a graph has bounded treewidth then the corresponding tree decomposition can be constructed in linear time [18].

Graphs on Surfaces We refer to the textbook "Graphs on Surfaces" by Mohar and Thomassen [110] for an introduction to the topological graph theory used here. Let $\mathcal{G}_g$ be the class of all graphs that can be embedded into a surface Σ of Euler-genus at most g. We say that a graph G is Σ-embedded if it is accompanied with an embedding of the graph into Σ. The *radial distance* between x and y is defined to be one less than the minimum length of a sequence starting from x and ending at y such that vertices and faces alternate in the sequence. Given an Σ-embedded graph $G = (V, E)$ and a set $S \subseteq V$, we denote by $\mathbf{R}^r_G(S)$ and $\mathbf{B}^r_G(S)$ the set of all vertices that are in radial distance at most r and distance at most r from some vertex in S respectively. Notice that for every set $S \subseteq V$ and every $r \geq 0$, it holds that $\mathbf{B}^r_G(S) \subseteq \mathbf{R}^{2r+1}_G(S)$ for any embedding of G into a surface Σ. An alternative way of viewing radial distance is to consider the *radial graph*, R_G: an embedded multigraph whose vertices are the vertices and the faces of G (each face f of G is represented by a point v_f in it). An edge between a vertex v and a vertex v_f is drawn if and only if v is incident to f. Thus R_G is a bipartite multigraph, embedded in the same surface as G. Hence, if $G \in \mathcal{G}_g$ then $R_G \in \mathcal{G}_g$. Also the radial distance between a pair of vertices in G corresponds to the normal distance in R_G.

t-Boundaried Graphs We define the notion of *t-boundaried graphs* and various operations on them.

Definition 1.2.2 [**t-Boundaried Graphs**] *A t-boundaried graph is a graph $G = (V, E)$ with t distinguished vertices, uniquely labelled from 1 to t. The set $\partial(G)$ of labelled vertices is called the boundary of G. The vertices in $\partial(G)$ are referred to as boundary vertices or terminals.*

For a graph $G = (V, E)$ and a vertex set $S \subseteq V$, we will sometimes consider the graph $G[S]$ as the $|\partial(S)|$-boundaried graph with $\partial(S)$ being the boundary.

Definition 1.2.3 [**Gluing by $\oplus$**] *Let G_1 and G_2 be two t-boundaried graphs. We denote by $G_1 \oplus G_2$ the t-boundaried graph obtained by taking the disjoint union of G_1 and G_2 and identifying each vertex of $\partial(G_1)$ with the vertex of $\partial(G_2)$ with the same label; that is, we glue them together on the boundaries. In $G_1 \oplus G_2$ there is an edge between two labelled vertices if there is an edge between them in G_1 or in G_2.*

Definition 1.2.4 [**Legality**] *Let $\mathcal{G}$ be a graph class, G_1 and G_2 be two t-boundaried graphs, and $G_1, G_2 \in \mathcal{G}$. We say that $G_1 \oplus G_2$ is legal with respect to $\mathcal{G}$ if the unified graph $G_1 \oplus G_2 \in \mathcal{G}$. If the class $\mathcal{G}$ is clear from the context we do not say with respect to which graph class the operation is legal.*

Definition 1.2.5 [**Replacement**] *Let $G = (V, E)$ be a graph containing an extended r-protrusion X. Let X' be the restricted protrusion of X and let G_1 be an r-boundaried graph. The act of replacing X' with G_1 corresponds to changing G into $G[V \setminus X'] \oplus G_1$. Replacing $G[X]$ with G_1 corresponds to replacing X' with G_1.*

Finite Integer Index

Definition 1.2.6 *For a parameterized problem, Π on a graph class $\mathcal{G}$ and two t-boundaried graphs G_1 and G_2, we say that $G_1 \equiv_\Pi G_2$ if there exists a constant c such that for all t-boundaried graphs G_3 and for all k,*

- *$G_1 \oplus G_3$ is legal if and only if $G_2 \oplus G_3$ is legal.*
- *$(G_1 \oplus G_3, k) \in \Pi$ if and only if $(G_2 \oplus G_3, k + c) \in \Pi$.*

Definition 1.2.7 [**Finite Integer Index**] *Π has finite integer index in $\mathcal{G}$ if for every t there exists a finite set $\mathcal{S}$ of t-boundaried graphs such that $\mathcal{S} \subseteq \mathcal{G}$ and for any t-boundaried graph G_1 there exists a $G_2 \in \mathcal{S}$ such that $G_2 \equiv_\Pi G_1$. Such a set $\mathcal{S}$ is called a set of representatives for (Π, t)*

Note that for every t, the relation $\equiv_\Pi$ on t-boundaried graphs is an equivalence relation. A problem Π is finite integer index, if and only if for every t, $\equiv_\Pi$ is of finite index, that is, has a finite number of equivalence classes. The term *finite integer index* first appeared in the work by Bodlaender and van Antwerpen-de Fluiter [25, 41] and is similar to the notion of *finite state* [2, 27, 38].

Digraphs A *digraph* or *directed graph* D is a set $V(D)$ of vertices and a set $A(D)$ of ordered pairs of vertices called *arcs*. Given a subset $V' \subseteq V(D)$ of a digraph D, by $D[V']$ we mean the digraph induced on V'. A vertex y of D is an *in-neighbor* (*out-neighbor*) of a vertex x if $yx \in A$ ($xy \in A$). The *in-degree* (*out-degree*) of a vertex x is the number of its in-neighbors (out-neighbors) in D. A *path* in a digraph is a sequence $P = p_1p_2 \ldots p_\ell$ of vertices such that $p_ip_{i+1} \in A(D)$ for every i. Let $P = p_1p_2 \ldots p_\ell$ be a given path. Then by $P[p_ip_j]$ we denote a subpath of P starting at vertex p_i and ending at vertex p_j. A *cycle* in D is a sequence $C = c_1c_2 \ldots c_\ell$ of vertices such that $c_ic_{i+1} \in A(D)$ for every i and $c_\ell c_1 \in A(D)$.

A digraph D is *acyclic* if D does not contain any cycles. It is well known that a digraph D is acyclic if and only if the vertices of D can be ordered into

$v_1 \dots v_n$ such that for every arc $v_iv_j \in A(D)$ we have $i < j$. Such an ordering is called a *topological* ordering of D. A *feedback arc set* of a digraph D is an arc set S such that $D \setminus S$ is acyclic. An *out-tree* is an acyclic digraph where every vertex except one has indegree 1 and one vertex, called the *root* with indegree 0. An *out-branching* in a digraph D is an out-tree in D containing all vertices of D. For a given vertex $q \in V(D)$, by *q-out-branching* (or *q-out-tree*) we denote an out-branching (out-tree) of D rooted at vertex q.

We say that the removal of an arc uv (or a vertex set S) *disconnects* a vertex w from the root r if every path from r to w in D contains arc uv (or one of the vertices in S). An arc uv is contracted as follows: add a new vertex u', and for each arc wv or wu add the arc wu', and for an arc vw or uw add the arc $u'w$, remove all arcs incident to u and v and the vertices u and v.

Let T be an out-tree of a digraph D. We say that u is a *parent* of v and v is a *child* of u if $uv \in A(T)$. We say that u is an *ancestor* of v if there is a directed path from u to v in T. An arc uv in $A(D) \setminus A(T)$ is called a *forward* arc if u is an ancestor of v, a *backward* arc if v is an ancestor of u and a *cross* arc, otherwise.

A *tournament* is a digraph T where every pair of vertices is connected by exactly one arc. An *arc weighted* tournament is a tournament which comes with a weight function $w : A \to \mathbb{R}$. For an arc weighted tournament we define the weight function $w^* : V \times V \to \mathbb{R}$ such that $w^*(u,v) = w(uv)$ if $uv \in A$ and 0 otherwise. Given a directed graph $D = (V, A)$ and a set F of arcs in A define $D\{F\}$ to be the directed graph obtained from D by reversing all arcs of F. The following is a useful characterization of minimal feedback arc sets in directed graphs.

Proposition 1.2.8 ([119]) *Let $D = (V, A)$ be a directed graph and F be a subset of A. Then F is a minimal feedback arc set of D if and only if F is a minimal set of arcs such that $D\{F\}$ is a directed acyclic graph.*

Given a minimal feedback arc set F of a tournament T, the ordering σ corresponding to F is the unique topological ordering of $T\{F\}$. Conversely, given an ordering σ of the vertices of T, the feedback arc set F corresponding to σ is the set of arcs whose endpoint appears before their startpoint in σ. The cost of an arc set F is $\sum_{e \in F} w(e)$ and the cost of a vertex ordering σ is the cost of the feedback arc set corresponding to σ.

Types of Logic The syntax of Monadic Second Order Logic (MSO_2) of graphs includes the logical connectives $\vee$, $\wedge$, $\neg$, $\Leftrightarrow$, $\Rightarrow$, variables for vertices, edges, set of vertices and sets of edges, the quantifiers $\forall$, $\exists$ that can be applied to these variables, and the following five binary relations: (1) $u \in U$ where u is a vertex variable and U is a vertex set variable; (2) $d \in D$ where d is an edge variable and D is an edge set variable; (3) $\mathbf{inc}(d, u)$, where d is an edge variable, u is a vertex variable, and the interpretation is that the edge d is incident on the vertex u; (4) $\mathbf{adj}(u, v)$, where u and v are vertex variables u, and the interpretation is that u and v are adjacent; (5) equality of variables representing vertices, edges, set of vertices and set of edges. If we do not have variables for sets of edges, this type of logic is called MSO_1 logic. If we only allow variables for vertices and edges, i.e there are no set variables we get First Order (FO) logic. An extention of MSO_2 called *counting monadic second-order logic* or CMSO is obtained from MSO_2 by also allowing atomic formulas for testing whether the cardinality of a set is equal to n modulo p, where n and p are integers such that $0 \leq n < p$ and $p \geq 2$. Essentially, CMSO is just MSO_2 equipped with the following atomic formula: If U denotes a set X, then $\mathbf{card}_{n,p}(U) = \mathbf{true}$ if and only if $|X|$ is n $\mathbf{mod}$ p. It is known that every set $\mathcal{F}$ of graphs of bounded treewidth is CMSO-definable if and only if $\mathcal{F}$ is finite state [101].

Vectors For a pair of integer row vectors $\hat{p} = [p_1, \ldots, p_t]$, $\hat{q} = [q_1, \ldots, q_t]$ we say that $\hat{p} \leq \hat{q}$ if $p_i \leq q_i$ for all i. The transpose of a row vector $\hat{p}$ is denoted by $\hat{p}^\dagger$. The t-sized vector $\hat{e}$ is $[1, 1, \ldots, 1]$, $\hat{0}$ is $[0, 0, \ldots, 0]$ and $\hat{e}_i$ is the t-sized vector with all entries 0 except for the i'th which is 1. Let $\tilde{O}(\sqrt{k})$ denote, as usual, any function which is $O(\sqrt{k}(\log k)^{O(1)})$. For any positive integer m put $[m] = \{1, 2, \ldots, m\}$.

Part I

Overview of the Field

Chapter 2

Algorithmic Techniques

2.1 Bounded Search Trees

Bounded Search Trees is a simple yet powerful technique that has found many applications in Parameterized Algorithm Design [116]. The basic idea is to reduce a given instance (I, k) to $f(k)$ independent instances $(I_1, k_1), \ldots (I_{f(k)}, k_{f(k)})$ in polynomial time such that

- For every $1 \leq i \leq f(k)$ we have $k_i < k$ and $|I_i| \leq |I| \cdot g(k)$ for a function g.
- Instances $(I, 0)$ are polynomial time solvable.
- The functions f and g are non-decreasing and depend only on k.

While the definition above might seem somewhat involved, most algorithms that apply the Bounded Search Tree technique do so with $f(k) \leq k^{o(1)}$ and $g(k) = 1$. One might even argue that usually $f(k)$ is a constant independent of k. Let (I, k) be the considered instance. If $k > 0$ the algorithm performs the reduction to $(I_1, k_1), \ldots (I_{f(k)}, k_{f(k)})$ in polynomial time and makes recursive calls to (I_i, k_i) for every i between 1 and $f(k)$. Let c be a constant such that reducing an instance to the $f(k)$ smaller ones and solving instances $(I, 0)$ with $|I| \leq n$ both take $O(n^c)$ time, and let $T(n, k)$ be an upper bound on the running time of the algorithm on an instance (I, k). Then $T(n, k) \leq f(k) \cdot T(g(k)n, k-1) + O(n^c)$. Expanding this recurrence k times yields $T(n, k) = O((f(k)g(k)^c)^k \cdot n^c))$. As an example we demonstrate how to apply this technique to get a simple FPT algorithm for the Vertex Cover problem where we are given a gragh G and an integer k and asked whether there is a subset S of $V(G)$ of size at most k such that every edge in $E(G)$ has at least one endpoint in S.

Theorem 2.1.1 ([108, 52]) *There is an $O(2^k n)$-time algorithm for* Vertex Cover.

Proof. First observe that a graph G has a vertex cover of size 0 if and only if G has no edges. Consider now an instance (G, k) of Vertex Cover with $k > 0$ and an edge $uv \in E(G)$. Any vertex cover S of G contains either u or v since S must contain at least one endpoint of uv. A set S that contains a vertex x is a vertex cover of G if and only if $S \setminus \{x\}$ is a vertex cover of $G \setminus x$ since the vertex x covers all edges incident to it. Hence G has a vertex cover of size at most k if and only if $G \setminus u$ or $G \setminus v$ has a vertex cover of size at most $k-1$. Thus there is a polynomial time reduction from the instance (G, k) to the two instances $(G \setminus u, k-1), (G \setminus v, k-1)$. The functions f and g in the discussion above are $f(k) = 2$ and $g(k) = 1$ and hence this algorithm runs in time $O(2^k \cdot n + m)$. ∎

The above algorithm first appeared already in 1984 in a monograph by Melhorn [108]. However, this went unnoticed by the FPT community. In the "Parameterized Complexity" monograph of Downey and Fellows [52], which was published in 1999, the history of the development of algorithms for the Vertex Cover problem is described. In 1987, Fellows and Langston showed that as a consequence of deep theorems in the Graph Minors project of Robertson and Seymour, Vertex Cover can be solved in $O(n^3)$ time for every fixed value of k [61]. Later the same year, Johnson showed that the Vertex Cover problem can be solved in time $O(n^2)$ for every fixed value of k. Neither Fellows and Langston nor Johnson give explicitely the dependence of the running time on k, and this dependence is at least doubly exponential in k. In 1988, Fellows [59] independently rediscovered the $O(2^k n + m)$ algorithm from Theorem 2.1.1. In 1989, Buss [52] described an algorithm with running time $O(kn + 2^k k^{2k+2})$. One should notice that in this algorithm, the dependence on k is worse than in the algorithm of Theorem 2.1.1, but that this dependence is separated completely from the dependence on n, that is, this term is additive instead of multiplicative. In 1992, combining the ideas of Buss and the algorithm in [59], Balasubramanian et al. [13] gave an algorithm with running time $O(kn + 2^k k^2)$. In 1996, Balasubramanian et al. [13] gave an algorithm with running time $O(kn + (\frac{4}{3})^k k^2)$. Independently of this development, in 1993 Papadimitriou and Yannakakis [118] gave an algorithm with running time $O(3^k n)$, based on matching techniques. In a series of papers, the ideas in [13] were refined, and to this date the best known algorithm for Vertex Cover is due to Chen et al. [30] and takes $O(1.274^k + kn)$ time. In the next section we give a brief introduction to some ideas that can be used to improve the running time of bounded search tree algorithms.

2.2 Reduction Rules and Branching Recurrences

A reduction rule is a rule or a polynomial time algorithm that transforms an instance (I, k) of a parameterized problem to an "equivalent and simpler" instance (I', k'). Here equivalent means that (I, k) is a yes instance if and only if (I', k')

is and the precise definition of what "simpler" means varies from problem to problem. It could mean that $|I'| < |I|$, it could mean that $k' < k$ or it could mean that the instance I' contains fewer occurances of a particular substructure. We will say that a reduction rule is *correct* if the instances (I, k) and (I', k') indeed are equivalent.

A good thing about reduction rules is that as long as one is only interested in whether (I, k) is a yes or no instance, there is absoutely no reason for not performing the reduction rule. Even better, quite often we can give performance guarantees on our reduction rules and show that an instance (I, k) to which our reduction rules can not be applied must satisfy $|I| \leq f(k)$ for some function f. In this case we say that the problem admits an $f(k) -$ *kernel*. The study of kernels has developed to become a flourishing subfield of Parameterized Algorithms and Complexity. A more thorough introduction to kernelization is deferred to Chapter 4. Reduction rules can often be useful independently of whether they give a kernel for the problem. In particular combining reduction rules with the Bounded Search Tree technique often gives improved running time bounds over a plain Bounded Search Tree algorithm. We illustrate this by describing an algorithm for Vertex Cover with running time better than the algorithm from Theorem 2.1.1.

Theorem 2.2.1 (Folklore) *There is an $O(1.6181^k n^2)$ time algorithm for Vertex Cover.*

Proof. We apply the following two rules: If G has a vertex u with degree 0 then let $(G', k') = (G \setminus u, k)$. If G has a vertex u with degree 1, let $(G', k') = (G \setminus N[u], k-1)$. Correctness of the first rule is obvious, and the second rule is correct because any vertex cover containing u could be swapped with a vertex cover containing $N(u)$ instead. If neither rule can be applied, all vertices have degree at least 2. Pick a vertex u. Any vertex cover of G must contain either u or $N(u)$. Thus (G, k) is a yes instance to Vertex Cover if and only if $(G \setminus u, k-1)$ or $(G \setminus N(u), k - |N(u)|)$ is. Let $T(n, k)$ is an upper bound on the running time of our algorithm. Since $|N(u)| \geq 2$ we have $T(n, k) \leq T(n, k-1) + T(n, k-2) + O(n^2)$. Thus $T(n, k) \leq O(\lambda^k n^2)$ where λ is the largest root of the characteristic polynomial $\lambda^k - \lambda^{k-1} - \lambda^{k-2}$ of the recurrence. The largest root of this polynomial is the golden ratio $(1 + \sqrt{5})/2 \leq 1.6181$ so $T(n, k) \leq O(1.6181^k n^2)$ completing the proof. ∎

A trait of branching algorithms is that the running time bound often can be improved by a refined case analysis. The positive aspect of this is that quite good running time bounds are achievable. A drawback is that considering more and more cases in the algorithm reduces clarity and makes the constant hidden in the big-Oh notation larger. Simulations have shown that in practice, the more advanced reduction and branching rules often slow the algorithm down [128].

2.3 Iterative Compression

Iterative Compression is a tool that has recently been used successfully to give FPT algorithms for a number of problems. This technique was first introduced by Reed, Smith and Vetta in order to solve the Odd Cycle Transversal problem [120]. In this problem we are given a graph G together with an integer k. The objective is to find a set S of at most k vertices whose deletion makes the graph bipartite, and a set S such that $G \setminus S$ is bipartite is called an odd cycle transversal of G. The method of Iterative Compression was used in obtaining faster fixed parameter tractable (FPT) algorithms for Feedback Vertex Set, Edge Bipartization, Chordal Deletion and Cluster Vertex Deletion on undirected graphs [42, 77, 107, 85]. The technique was also used by Chen et al. [31] to show that the Directed Feedback Vertex Set problem is FPT, resolving a long standing open problem in Parameterized Complexity. In this section we present a reinterpretation of the algorithm given by Reed, Smith and Vetta for Odd Cycle Transversal. In particular, we give a new proof of the following theorem.

Theorem 2.3.1 *There is an algorithm that given a graph $G = (V, E)$ and integer k decides whether G has an odd cycle transversal of size at most k in time $O(3^k \cdot k \cdot |E| \cdot |V|)$.*

Proof. The idea is to reduce the problem in question to a modified version, where we are also given as input a solution that is almost good enough, but not quite. For the case of Odd Cycle Transversal, we are given an odd cycle transversal S' of G of size $k + 1$. We call this problem the *compression* version of Odd Cycle Transversal. The crux of the Iterative Compression method is that often the compression version of a problem is easier to solve than the original one.

Suppose we could solve the compression version of the problem in $O(f(k)n^c)$ time. We show how to solve the original problem in $O(f(k)n^{c+1})$ time. Order the vertices of $V(G)$ into $v_1v_2 \ldots v_n$ and define $V_i = \{v_1 \ldots v_i\}$ for every i. Notice that if G has an odd cycle transversal S of size k then $S \cap V_i$ is an odd cycle transversal of $G[V_i]$ for every $i \leq n$. Furthermore, if S is an odd cycle transversal of $G[V_i]$ then $S \cup \{v_{i+1}\}$ is an odd cycle transversal of $G[V_{i+1}]$. Finally, V_k is an odd cycle transversal of size k of $G[V_k]$. These three facts together with the $f(k)n^c$ algorithm for the compression version of Odd Cycle Transversal give an $f(k)n^{c+1}$ time algorithm for Odd Cycle Transversal as follows. Call the algorithm for the compression version with input $(G[V_{k+1}], V_{k+1}, k)$. The algorithm will either report that $(G[V_{k+1}, k])$ has no odd cycle transversal of size k or return such an odd cycle transversal, call it S_{k+1}. In the first case G has no odd cycle transversal of size k. In the second, call the algorithm for the compression version with input $(G[V_{k+2}], S_{k+1} \cup \{v_{k+2}\}, k)$. Again we either receive a “no” answer or a odd cycle transversal S_{k+2} of $G[V_{k+2}]$ of size k and again, if the answer is negative then G has no k-sized odd cycle transversal. Otherwise we call the compression algorithm with input $(G, S_{k+2} \cup \{v_{k+3}\}, k)$ and keep going on in a similar manner. If we receive a negative answer at some step we answer that G has no k-sized odd cycle transversal. If we do not receive a negative answer at any step, then after $n-k$ calls to the compression algorithm we have a k-sized odd cycle transversal of $G[V_n] = G$. Thus we have resolved the input instance in time $O(f(k)n^{c+1})$. We refer to [116] for a more thorough introduction to Iterative Compression.

We now show how to solve the compression version of Odd Cycle Transversal in time $O(3^k \cdot k \cdot |E|)$. From the discussion in the previous paragraph it will follow that Odd Cycle Transversal can be solved in time $O(3^k \cdot k \cdot |E| \cdot |V|)$. For two vertex subsets V_1 and V_2 of $V(G)$ a walk from V_1 to V_2 is a walk with one endpoint in V_1 and the other in V_2, or a single vertex in $V_1 \cap V_2$. The following is a simple fact about bipartite graphs.

Fact 2.3.2 *Let $G = (V_1 \uplus V_2, E)$ be a bipartite graph with vertex bipartition $V_1 \uplus V_2$. Then*

1. *For $i \in \{1,2\}$, no walk from V_i to V_i has odd length.*
2. *No walk from V_1 to V_2 has even length.*

Recall that we are given a graph G and an odd cycle transversal S' of G of size $k+1$ and we have to decide whether G has an odd cycle transversal of size at most k. If such an odd cycle transversal S exists then there exists a partition of S' into $L \uplus R \uplus T$, where $T = S' \cap S$ and L and R are subsets of the left and right bipartitions of the resulting graph. The algorithm iterates over all 3^k partitions of S into $L \uplus R \uplus T$. For each partition we run an algorithm that takes as input a partition of S' into $L \uplus R \uplus T$, runs in $O(k \cdot |E|)$ time and decides whether there exists a set of vertices T' of size at most $k - |T|$ in $G \setminus S'$ such that $G \setminus (T \cup T')$ is bipartite with bipartitions V_L and V_R such that $L \subseteq V_L$ and $R \subseteq V_R$. In the remainder of this section we give such an algorithm. This algorithm together with the outer loop over all partitions of S' yields the $O(3^k \cdot k \cdot |E|)$ time algorithm for the compression step.

Before proceeding we do a simple "sanity check". If there is an edge in $G[L]$ or $G[R]$ it is clear that X can not exist since then either V_L or V_R can not be an independent set. Hence if there is an edge in $G[L]$ or $G[R]$ we can immediately skip to the next partition of S'. Now, since $G \setminus S'$ is bipartite, let $A \uplus B$ be a bipartition of $G \setminus S'$. Let A_l and B_l be the neighbors of L in A and B respectively. Similarly let A_r and B_r be the neighbours of R in A and B respectively.

Claim 2.3.3 *Let (G, S', k) be an instance of the compression version of* Odd Cycle Transversal *and let $S' = L \uplus R \uplus T$. If $X \subseteq (V(G) \setminus S')$ is a set of* *vertices such that $G \setminus (T \cup X)$ is bipartite with bipartitions V_L and V_R such that $L \subseteq V_L$ and $R \subseteq V_R$, then in $G \setminus (S' \cup X)$, there are no paths between A_l and B_l; B_l and B_r; B_r and A_r; and, A_r and A_l.*

Proof. Any path from A_l to B_l in $G \setminus (S' \cup X)$ has odd length and can be extended to a walk from L to L of odd length in $G' \setminus (T \cup X)$, contradicting Fact 2.3.2. A symmetric argument shows that there are no paths between B_r and A_r in $G \setminus (S' \cup X)$. Any path from B_l to B_r in $G \setminus (S' \cup X)$ must be of even length and can be extended to a walk in $G \setminus (T \cup X)$ from L to R of even length, again contradicting Fact 2.3.2. A symmetric argument yields that there are no paths between A_r and A_l. ■

Claim 2.3.4 *Let (G, S', k) be an instance of the compression version of* Odd Cycle Transversal *and let $S = L \uplus R \uplus T$ such that $G[L]$ and $G[R]$ are independent sets. Let X be a set of vertices in $V(G) \setminus S'$ such that in $G \setminus (S' \cup X)$, there are no paths between A_l and B_l; B_l and B_r; B_r and A_r; and, A_r and A_l. Then $G \setminus (T \cup X)$ is bipartite with bipartitions V_L and V_R such that $L \subseteq V_L$ and $R \subseteq V_R$.*

Proof. Notice that every path from a vertex in L to another vertex in L with inner vertices only in $V(G) \setminus (S' \cup X)$ must have even length. Similarly every path from a vertex in R to another vertex in R with inner vertices only in $V(G) \setminus (S' \cup X)$ must have even length and every path from a vertex in L to a vertex in R with inner vertices only in $V(G) \setminus (S' \cup X)$ must have odd length. Since $G[L]$ and $G[R]$ are independent sets it follows that if $G \setminus (T \cup X)$ is bipartite then it has bipartitions V_L and V_R such that $L \subseteq V_L$ and $R \subseteq V_R$. We now prove that $G \setminus (T \cup X)$ is bipartite.

Consider a cycle in $G \setminus (T \cup X)$. If C does not contain any vertices of $(L \cup R)$ then $|E(C)|$ is even since $G \setminus S'$ is bipartite. Let $v_1, v_2, \ldots v_t$ be the vertices of $(L \cup R) \cap C$ in their order of appearance along C. Let $v_0 = v_t$, then we have that $|E(C)| = \sum_{i=0}^{t-1} d_c(v_i, v_{i+1})$. But then $E(C)$ must be even since the number if indices i such that $v_i \in L$ and $v_{i+1} \in R$ is equal to the number of indices j such that $v_j \in R$ and $v_{j+1} \in L$. This concludes the proof. ■

To check whether $G \setminus T$ has an odd cycle transversal X such that $G \setminus (T \cup X)$ is bipartite with bipartitions V_L and V_R such that $L \subseteq V_L$ and $R \subseteq V_R$ we proceed as follows. Construct an auxiliary graph $\widetilde{G}$ from $G \setminus S'$ by introducing two special vertices s, t and connecting s to each vertex in $A_l \cup B_r$ and t to each vertex in $A_r \cup B_l$. The Claims 2.3.3 and 2.3.4 show that it is sufficient to check whether there is an st-separator in $\widetilde{G}$ of size at most $k - |T \cup T'|$. This can be done using max flow in time $O(k \cdot |E|)$. This discussion together with Claims 2.3.3 and 2.3.4 complete the proof of Theorem 2.3.1 ■

2.4 Dynamic Programming and Courcelle's Theorem

The Dynamic Programming (DP) technique has found many applications in the design of algorithms, Parameterized Algorithms being no exception. When designing Dynamic Programming based parameterized algorithms, the key is to try to keep the size of the DP table and the time needed to compute each cell in the table down to $f(k)n^{O(1)}$.

A well-known example of Dynamic Prorgramming in Parameterized Algorithms is the Dreyfus-Wagner algorithm for the Steiner Tree problem [54]. In the Steiner Tree problem we are given a connected graph G together with a subset X of $V(G)$. The set X is called a set of *terminals* and $|X| = k$. The objective is to find a subtree T of G containing all terminals, minimizing $|V(T)|$, the number of vertices in T. The Dreyfus-Wagner algorithm solves this problem in time $O(3^k n^{O(1)})$. Interestingly, if you parameterize Steiner Tree problem by the maximum number of *non-terminal* vertices allowed in the solution, the problem becomes intractable. This is discussed in detail in Section 3.3.

Parameterized problems where the parameter is the treewidth of the input graph G are often tackled by doing dynamic programming over the tree decomposition of G. In particular it has been shown that Independent Set, Vertex Cover and Dominating Set in graphs of treewidth at most k can be solved in time $O(2^k k^{O(1)} n)$, $O(2^k k^{O(1)} n)$ and $O(4^k n)$ respectively [116]. Many other problems admit fast dynamic programming algorithms in graphs of bounded treewidth.

A very useful tool for showing that a problem is FPT parameterized by treewidth is the celebrated Courcelle's theorem, which states that every problem expressible in Monadic Second Order Logic is fixed parameter tractable parameterized by the treewidth of the input graph. For a graph predicate ϕ expressed in MSO_2 let $|\phi|$ be the length of the MSO_2 expression for ϕ.

Theorem 2.4.1 (Courcelle's Theorem [35]) *There is a function $f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ and an algorithm that given a graph G together with a tree-decomposition of G of width t and a MSO_2 predicate ϕ decides whether $\phi(G)$ holds in time $f(|\phi|, t)n$.*

To apply Courcelle's theorem on a specific problem we need to show that the problem is expressible in monadic second order logic. For example consider the Independent Set problem. Here we are given as input a graph G, a tree-decomposition of G of width t and an integer k. The objective is to decide whether G has an independent set of size at least k. We formulate the Independent Set problem in MSO_2. That is, a graph G has an independent set of size k if and only if the following predicate holds:

$$\begin{aligned}\phi(G) &= \exists v_1, v_2, \ldots, v_k \in V(G) : v_1 \neq v_2 \wedge \neg\mathbf{adj}(v_1, v_2), v_1 \neq v_3 \wedge \neg\mathbf{adj}(v_1, v_2), \\ &\quad \ldots v_2 \neq v_3 \wedge \neg\mathbf{adj}(v_2, v_3), \ldots v_{k-1} \neq v_k \wedge \neg\mathbf{adj}(v_{k-1}, v_k)\end{aligned}$$

Observe that the length of the predicate ϕ depends only on k, and not on the size of the graph G. Hence, by Theorem 2.4.1, there is an algorithm that given a graph G together with a tree-decomposition of G of width at most t and an integer k decides whether G has an independent set of size at least k in time $f(k,t)n$ for some function f. In fact, it is not really necessary that the tree-decomposition of G is given. Due to a result of Bodlaender [18], a tree-decomposition of width t of a graph G of treewidth t can be computed in time $f(t)n$ for some function f.

Theorem 2.4.2 (Bodlaender's Theorem [18]) *There is a function $f : \mathbb{N} \rightarrow \mathbb{N}$ and an $f(t)n$ time algorithm that given a graph G and integer t decides whether G has treewidth at most t, and if so, constructs a tree-decomposition of width at most t.*

Hence, combining Theorems 2.4.1 and 2.4.2 yields that there is a function $f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ and an algorithm that given a graph G of treewidth t and a MSO_2 predicate ϕ decides whether $\phi(G)$ holds in time $f(|\phi|, t)n$. For example, by the discussion in the previous paragraph there is an algorithm that given a graph G of treewidth t and an integer k decides whether G has an independent set of size at most k in time $f(t,k)n$ for some function f.

Theorem 2.4.1 has been generalized even further. For instance it has been shown that MSO_2-*optimization* problems are fixed parameter tractable parameterized by the treewidth of the input graph. In a MSO_2-*minimization* problem you are given a graph G and a predicate ϕ in MSO_2 that describes a property of a vertex (edge) set in a graph. The objective is to find a vertex (edge) set S of minimum size such that $\phi(G,S)$ holds. In a MSO_2-*maximization* problem the objective is to find a set S of maximum size such that $\phi(G,S)$ holds.

Theorem 2.4.3 ([27, 12]) *There is a function $f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ and an algorithm that given a graph G of treewidth t and a MSO_2 predicate ϕ finds a largest (smallest) set S such that $\phi(G,S)$ holds in time $f(|\phi|, t)n$.*

Turning our attention back to the Independent Set problem, we can observe that by applying Theorem 2.4.3 we can obtain a stronger result than from Theorem 2.4.1. In particular, the Independent Set can be expressed as a MSO_2-maximization problem as follows:

$$\begin{aligned}&\max |S| \text{ s.t:} \\ &\quad \phi(G,S) = \forall v_1, v_2 v_1 = v_2 \vee \neg\mathbf{adj}(v_1, v_2) \text{ holds}\end{aligned}$$

Hence, by Theorem 2.4.3 there is an algorithm that given a graph G of treewidth t as input can find a maximum size independent set in time $f(t)n$ for some function f.

Another way to generalize Theorem 2.4.1 is to consider larger classes of graphs than graphs of bounded treewidth. In particular, if we restrict the predicate ϕ to MSO_1 logic, Theorem 2.4.3 can be extended to graphs of bounded cliquewidth.

Theorem 2.4.4 ([36]) *There is a function $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ and an algorithm that given a graph G and a clique-expression of G of width t and a MSO_1 predicate ϕ finds a largest (smallest) set S such that $\phi(G, S)$ holds in time $f(|\phi|, t)n$.*

There are natural problems expressible in MSO_2 but not in MSO_1. Examples include Hamiltonian Cycle and Max Cut. In Section 7.2 we show that up to certain complexity-theoretic assumptions, one can not hope to be able to generalize Theorem 2.4.4 to also handle all problems in MSO_2.

2.5 Well Quasi Ordering

The set of natural numbers is *well-ordered* since every two natural numbers are comparable, that is, for any two numbers a and b such that $a \neq b$, either $a < b$ or $b < a$. An unusual way to describe a well-ordered set is to say that it contains no *anti-chain* of length at least 2, where an anti-chain is a sequence $a_1, a_2, \ldots a_k$ such that for every $i \neq j$ we have $a_i \neq a_j$ and neither $a_i < a_j$ nor $a_i > a_j$. We can relax the notion of well-ordering, and say that a set S is *well-quasi-ordered* under a relation $<$ if every anti-chain in S is finite.

Robertson and Seymour, proved in their *graph minors* project that the set of graphs is well-quasi-ordered under the minor relation, thereby proving the *Graph Minor Theorem* and resolving *Wagner's Conjecture* [135]. On the way they defined many other interesting and useful concepts and showed results that have been very useful for Parameterized Algorithms and Complexity. The concept of *treewidth* was for instance introduced as a part of the graph minors project, and the Bidimensionality theory discussed in Section 2.6 relies heavily on the graph minors project. In this section, however, we only discuss the direct implications of the *Graph Minor Theorem* to Parameterized Algorithms and Complexity.

Proposition 2.5.1 (Graph Minor Theorem [122]) *The set of graphs is well-quasi-ordered under the minor relation.*

Consider a graph class $\mathcal{G}$ that is closed under taking minors. That is, if $G \in \mathcal{G}$ and $H \leq_M G$ then $H \in \mathcal{G}$ as well. Consider the set of graphs $\mathcal{F}$ consisting of all graphs *not* in $\mathcal{G}$ such that all their minors is in $\mathcal{G}$. We call this set the *set of forbidden minors* of $\mathcal{G}$. Notice that every graph G that is not in $\mathcal{G}$ must have some minor in $\mathcal{F}$. Notice also that by definition, $\mathcal{F}$ is an antichain, and that by the Graph Minor Theorem $\mathcal{F}$ is finite. Hence, to check whether G belongs to $\mathcal{G}$ it is sufficient to check for every $H \in \mathcal{F}$ whether G contains H as a minor. To this end, the following theorem, also from the graph minors project, is useful.

Proposition 2.5.2 ([121]) *For every fixed graph H there is an $O(n^3)$ time algorithm to check for an input graph G whether $H \leq_M G$.*

Combining the Graph Minors Theorem with Proposition 2.5.2 yields that every minor closed graph class $\mathcal{G}$ can be recognized in $O(n^3)$ time, where the constant hidden in the big-Oh notation depends on $\mathcal{G}$. To put this in terms of Parameterized Algorithms - the problem of deciding whether G belongs to a minor closed class $\mathcal{G}$ is FPT parameterized by $\mathcal{G}$.

Corollary 2.5.3 ([61]) Vertex Cover *is fixed parameter tractable.*

Proof. We prove that if a graph G has a vertex cover C of size at most k then so do all the minors of G. For any edge $e \in E(G)$, C is a vertex cover of $G \setminus e$. Similarly for any $v \in V(G)$, $C \setminus \{v\}$ is a vertex cover of $G \setminus v$. Finally, for an edge $uv \in E(G)$ let G_{uv} be the graph obtained by contracting the edge uv and let u' be the vertex obtained from u and v during the contraction. Then, every edge in $E(G)$ with u or v being one of its endpoints will have u' as its endpoint in G_{uv}. Hence, if neither u nor v are in C then C is a vertex cover of G_{uv}. If $u \in C$ or $v \in C$ then $(C \setminus \{u, v\}) \cup \{u'\}$ is a vertex cover of size at most k of G_{uv}. Hence, for a fixed integer $k \geq 0$ the class $\mathcal{C}_k$ of graphs that have a vertex cover of size at most k is closed under taking minors.

By Theorem 2.5.1 $\mathcal{C}_k$ has a finite set $\mathcal{F}_k$ of forbidden minors. By Proposition 2.5.2, for each $H \in \mathcal{F}_k$ we can decide whether G contains H as a minor in time $O(f(|H|)n^3)$ for some function $f : \mathbb{N} \to \mathbb{N}$. Let h be the size of the largest graph in $\mathcal{F}_k$. Then deciding whether a particular graph G is in $\mathcal{C}_k$ can be done in time $O(|\mathcal{F}_k| f(h) n^3)$. ■

For a fixed minor closed graph class $\mathcal{G}$, consider the following problem. Input is graph G and integer k. The parameter is k and the objective is to determine whether there exists a set vertex set S of size at most k such that $G \setminus S \in \mathcal{G}$. The proof of Corollary 2.5.3 can easily be modified to show that for any fixed minor closed graph class $\mathcal{G}$ this problem is FPT.

2.6 Win / Wins

The concept of Win/Wins is simple, yet it has proven useful both for showing problems fixed parameter tractable and for designing efficient FPT algorithms. A Win / Win consists of perfoming a polynomial time test on the input instance, and then proceeding depending on the test outcome. The idea is that for some tests all outcomes give useful information about the instance in question. We give an example with MAX LEAF SPANNING TREE (MLST). In this problem we are given a graph G together with an integer k and asked whether there is a spanning tree of G with at least k leaves.

Theorem 2.6.1 ([58]) MAX LEAF SPANNING TREE *is fixed parameter tractable.*

Proof. On an input instance (G, k) we do a polynomial test. In particular we pick an arbitrary vertex $v \in V(G)$ and run a breadth first search (BFS) starting at v. There are three outcomes, (a) : G is disconnected. (b) : G is connected, some BFS-layer has at least k vertices or (c) G is connected and all BFS layers have at most $k-1$ vertices. We show how all three outcomes give us information that help us decide the instance.

If G is disconnected then G has no spanning tree, so we can immediately answer no. Assume now that G is connected. If some BFS-layer has at least k vertices then the BFS-tree rooted at v is a spanning tree with at least k leaves and we can answer yes. If all BFS layers have at most $k-1$ vertices, we construct a path decomposition of G my making a bag out of every two consequtive layers. This decomposition has width at most $2k-3$ so the pathwidth of G is at most $2k-3$ in this case. Since the property that G has at least k leaves is expressible in monadic second order logic, the theorem follows by Courcelle's theorem. Expressing that G has at least k leaves in monadic second order logic is routine.
■

2.6.1 Bidimensionality

Win / Wins have been especially useful to develop subexponential time fixed parameter algorithms for problems in planar graphs, and more generally in graphs excluding a fixed graph H as a minor. For brevity, we only describe how these algorithms work in planar graphs. Many parameterized problems are optimization problems parameterzied by the objective function value. The objective function is said to be *bidimensional*, if the optimal value of an $n \times m$-grid is $O(nm)$, and if the optimal value of any minor H of G is at most the optimal value of G. A parameterized problem is *bidimensional* if the input is a graph G and an integer k which is the parameter, and the objective is to determine whether the optimal value of a bidimensional objective function on G is at most k. Most bidimensional problems exhibit nice algorithmic properties on planar graphs. In particular, it was proved by Demaine et al. [43] that any bidimensional problem which can be solved in time $2^{O(t+k)}n^{O(1)}$ on graphs of treewidth t admits a $2^{O(\sqrt{k})}n^{O(1)}$ time algorithm in planar graphs. The results proved in [43] are for more general classes of graphs and more general classes of problems than considered here. We demonstrate how these ideas apply to VERTEX COVER in planar graphs. We will need two propositions about about treewidth and graph minors that we will use as black boxes. Notice that if a minor of G has treewidth at least t then the treewidth of G is also at least t.

Proposition 2.6.2 (Excluded Grid Theorem [123]) *Every planar graph G of treewidth at least t contains a $\frac{t}{4} \times \frac{t}{4}$-grid as a minor.*

Proposition 2.6.3 ([125]) *The treewidth of planar graphs can be $\frac{3}{2}$-approximated in polynomial time.*

We explain how to exploit the above propositions to show that VERTEX COVER in planar graphs can be solved in time $2^{O(\sqrt{k})}$. As observed in Section 2.5, if a graph has a vertex cover of size at most k, then so do all its minors. Observe also that the size of a minimum vertex cover of a $k \times k$ grid is at least $k^2/2$.

Theorem 2.6.4 ([43, 46, 49]) *There is an $2^{O(\sqrt{k})}n^{O(1)}$ time algorithm for* VERTEX COVER *on planar graphs.*

Proof. On an input instance (G, k) we perform a test in polynomial time - we run the 3/2-approximation for treewidth on G, and let t be the width of the decomosition returned by the approximation algorithm. We have two possible outcomes, either $(\frac{2t}{3*4})^2/2 = \frac{t^2}{72} > k$ or not. If not, then we can find an optimal vertex cover in time $O(2^t n^{O(1)}) \leq 2^{O(\sqrt{(k)}}n^{O(1)}$ by applying the $2^t n^{O(1)}$ time dynamic programming algorithm for VERTEX COVER in graphs of bounded treewidth []. If $\frac{t^2}{72} > k$ then G contains a $(\sqrt{2k+1} \times \sqrt{2k+1})$ grid as a minor. The size of the minimum vertex cover of this grid is more than k and hence (G, k) is a no-instance. ∎

Algorithms of the above type can be given for any bidimensional problem which can be solved efficiently in grahps of bounded treewidth, and several survery papers have been written on Bidimensionality [43, 46, 49].

2.7 Color Coding

The Color Coding technique was developed by Alon et al [11] as a tool to detect a k-sized subgraph F of constant treewidth in an input graph G in time $2^{O(k)}n^{O(1)}$. The color coding technique is typically applied when we are searching for a small structure S in a larger structure X. We randomly color X with a set of colors and show that if X contains S as a substructure then the probability that S has been colored "nicely" is high. Here "nicely" means that the coloring is helpful for finding S quickly. In this section we describe the color coding algorithm of Alon et al [11] for the LONGEST PATH problem. In this problem we are given as input a graph G and an integer k and asked whether G contains a path on k vertices as a subgraph.

We start by giving a *randomized* algorithm for the problem, and then show how to *derandomize* it in order to get a deterministic algorithm. Our randomized algorithm will run in time $O((2e)^k n^{O(1)})$ and if G does not contain a k-path then the algorithm returns that it does not. On the other hand, of G does contain a k-path the algorithm detects such a path with probability at least 3/4. Repeating the algorithm more times gives an arbitrarily good probability of returning the correct answer.

The algorithm proceeds as follows. We color $V(G)$ with colors from $\{1, \ldots, k\}$ uniformly at random, and run an algorithm to detect whether there is a path P on k the vertices of P have been distinctly colored. We say that a path is *multicolored* if it contains at most one vertex of each color. If G contains a path P' on k vertices then the probability that P is multicolored is $k!/k^k = (1/e^k)k^{O(1)}$ by Stirlings formula.

Lemma 2.7.1 ([11]) *There is an $O(2^k n^{O(1)})$ time algorithm to decide whether there is a multicolored path in a colored graph G.*

Proof. Let $V_1, \ldots, V_k$ be a partitioning of $V(G)$ such that all vertices in V_i are colored i. We apply dynamic programming, for a subset S of $\{1, \ldots, k\}$ and a vertex $u \notin \bigcup_{i \in S} V_i$ we define the function $PATH(S, u)$ to return $TRUE$ if there is a multicolored path on $|S| + 1$ vertices in $G[\{u\} \bigcup_{i \in S} V_i]$ with one endpoint in u, and $FALSE$ otherwise. Since any multicolored path can visit every color class at most once the recurrence

$$PATH(S, u) = \bigvee_{i \in S, v \in V_i, uv \in E(G)} PATH(S \setminus \{i\}, v)$$

holds. Clearly all values of $PATH$ can be computed in $O(2^k n^{O(1)})$ time by applying the above recurrence. To determine whether there is a multicolored k-path in G we loop over every i from 1 to k and for each vertex in V_i check whether $PATH(v, \{1, \ldots, i-1, i+1, \ldots, k\})$ is set to $TRUE$. If it is for some choice for i and v, then there is a path, otherwise not. Given a positive answer the multicolored path itself can be reconstructed using a standard backtracking technique. ∎

Suppose G has a k-path. If we color $V(G)$ with colors from $\{1, \ldots, k\}$ uniformly at random and then run the algorithm from Lemma 2.7.1 then the probability that we detect this path is $(1/(e^k k^{O(1)})$. On the other hand, if G has no k-path then we correctly return that it indeed does not. If we repeat this procedure $O(e^k k^{O(1)})$ times the probability that we detect a k-path, if it exists, is at least 3/4. This yields the $O((2e)^k n^{O(1)})$ expected time algorithm for the Longest Path problem.

We now discuss how to *derandomize* the algorithm, that is, to replace the randomized coloring step by a deterministic step that does the same job. To this end, the following result is helpful.

Proposition 2.7.2 ([114]) *For every n, k there is a family of functions $\mathcal{F}$ of size $O(e^k \cdot k^{O(\log k)} \cdot \log n)$ such that every function $f \in \mathcal{F}$ is a function from $\{1, \ldots, n\}$ to $\{1, \ldots, k\}$ and for every subset S of $\{1, \ldots, n\}$ there is a function $f \in \mathcal{F}$ that is bijective when restricted to S. Furthermore, given n and k, $\mathcal{F}$ can be computed in time $O(e^k \cdot k^{O(\log k)} \cdot \log n)$.*

The family $\mathcal{F}$ above can replace the random coloring step of our algorithm. In particular, instead of the random coloring step we construct the family $\mathcal{F}$, iterate over every $f \in \mathcal{F}$ and use f as a coloring of the graph. It follows from Proposition 2.7.2 that if G contains a path P on k vertices then there is some $f \in \mathcal{F}$ such that P is multicolored when f is used to color $V(G)$. This yields the following theorem.

Theorem 2.7.3 ([11, 114]) *There is an $O((2e)^k n^{O(1)})$ time algorithm for* LONGEST PATH.

For the case of LONGEST PATH, algorithms with improved running times have later been published. The currently fastest deterministic algorithm for the problem runs in time $O(4^k n^{O(1)})$ and is based on a combination of the Color Coding and Divide and Conquer techniques [95]. The fastest randomized algorithm is due to Williams [136] and runs in expected time $O(2^k n^{O(1)})$.

2.8 Integer Linear Programming

In the early 80's, Lenstra [102] proved that INTEGER LINEAR PROGRAMMING is Fixed Parameter Tractable when parameterized by the number p of variables, giving an algorithm with running time doubly exponential in p. His result was subsequently improved by Kannan [88] to a $p^{O(p)} n^{O(1)}$ time algorithm. This algorithm uses Minkowski's Convex Body theorem and other results from Geometry of Numbers. A bottleneck in this algorithm was that it required space exponential in p. Using the method of simultaneous Diophantine approximation, Frank and Tardos [71] describe preprocessing techniques, using which it is shown that Lenstra's and Kannan's algorithms can be made to run in polynomial space. They also slightly improve the running time of the algorithm. For our purposes, we will use this algorithm.

Because of the vast expressive power of Integer Linear Programming it is natural to expect that it should be possible to prove problems Fixed Parameter Tractable by reducing the problem to INTEGER LINEAR PROGRAMMING with few variables. Quite surprisingly this technique has not yet found many applications in Parameterized Complezity. To the authors best knowledge the method has only been used to give a FPT algorithm for the CLOSEST STRING problem [75] and, in an EPTAS for MIN-MAKESPAN-SCHEDULING problem [113] and to prove that four graph layout problems are classFPT when parameterized by the size of the minimum vertex cover of the input graph [63]. We believe that it is quite probable that Integer Linear Programming could turn out useful for other problems as well. In this section we state the theorems necessary to apply the method and give an example of an FPT algorithm based on INTEGER LINEAR PROGRAMMING.

> INTEGER LINEAR PROGRAMMING (p-ILP): Given matrices $A \in \mathbb{Z}^{m\times p}$ and $b \in \mathbb{Z}^{m\times 1}$, the question is whether there exists a vector $\bar{x} \in \mathbb{Z}^{p\times 1}$ satisfying the m inequalities, that is, $A \cdot \bar{x} \leq b$. The number of variables p is the parameter.

Theorem 2.8.1 ([88],[102],[71]) p-ILP *can be solved using* $O(p^{2.5p+o(p)} \cdot L)$ *arithmetic operations and space polynomial in* L. *Here* L *is the number of bits in the input and* p *the number of variables in the integer linear program.*

In order to be able to use an algorithm for p-ILP as a subroutine in our algorithms, we need the optimization version of p-ILP rather than the feasibility version. We proceed to define the minimization version of p-ILP.

> p-VARIABLE INTEGER LINEAR PROGRAMMING OPTIMIZATION (p-OPT-ILP): Let matrices $A \in \mathbb{Z}^{m\times p}$, $b \in \mathbb{Z}^{m\times 1}$ and $c \in \mathbb{Z}^{1\times p}$ be given. We want to find a vector $\bar{x} \in \mathbb{Z}^{p\times 1}$ that **minimizes** the objective function $c \cdot \bar{x}$ and satisfies the m inequalities, that is, $A \cdot \bar{x} \geq b$. The number of variables p is the parameter.

Theorem 2.8.2 p-OPT-ILP *can be solved using* $O(p^{2.5p+o(p)} \cdot L \cdot \log(MN))$ *arithmetic operations and space polynomial in* L. *Here,* L *is the number of bits in the input,* N *is the maximum of the absolute values any variable can take, and* M *is an upper bound on the absolute value of the minimum taken by the objective function.*

Proof. We can first do a binary search to find the minimum value of the objective function. For an example suppose we guess that $c \cdot \bar{x} \leq \frac{M}{2}$. Now we make a new matrix A' of dimension $(m+1) \times p$ whose first row consists of c and rest of the rows is $-A$(by multiplying each entry of A by -1). We will denote the matrix A' as $[\frac{c}{-A}]$. Similarly we make $b' = [\frac{M/2}{-b}]$. Now we apply Theorem 2.8.1 to check whether there exists a vector $x' \in \mathbb{Z}^{p\times 1}$ such that $A' \cdot x' \leq b'$. Having found the minimum value of the objective function in this way, we find the lexicographically smallest solution satisfying these inequalities. We determine the value of one variable at a time by doing a binary search similar to the one we used to find the minimum of the objective function. For all this we need to run the algorithm for ILP easibility at most $O(p \cdot \log N + \log M)$ times. ■

Now we give an example for how Theorem 2.8.2 can be used applied to show that a problem is FPT.

An Algorithm for Graph Imbalance: In order to define the IMBALANCE problem we need to introduce some notation. A permutation $\pi : V \to \{1, 2, \ldots, n\}$ orders the vertex set into $v_1 <_\pi v_2 <_\pi \ldots <_\pi v_n$. For every i, the set V_i is $\{v_1, \ldots, v_i\}$ and $\partial(V_i) = \{uv \mid uv \in E, u \in V_i, v \in V \setminus V_i\}$. We define $L_\pi(v)$ to be $\{u \mid u \in N(v), u <_\pi v\}$ and $R_\pi(v)$ is $\{u \mid u \in N(v), v <_\pi u\}$.

In the IMBALANCE problem, we are given a graph $G = (V, E)$ with a vertex cover $C = \{c_1, \ldots, c_k\} \subseteq V$ of size k as input and asked to find a permutation $\pi : V \to \{1, 2, \ldots, n\}$ that minimizes $\sum_{i \le n} |L_\pi(v_i) - R_\pi(v_i)|$. The parameter is k, that is, the size of the vertex cover C. One should notice that this means that an algorithm whose running time depends exponentially on the objective function is *not* and FPT algorithm for this parameterized problem. We show how to apply Theorem 2.8.2 in order to give an FPT algorithm for IMBALANCE parameterized by the size of the minimum vertex cover of G.

We are looking for a permutation $\pi : V \to \{1, 2, \ldots, n\}$ for which $f_{im}(\pi)$ is minimized. In order to do this, we loop over all possible permutations of the vertex cover C and for each such permutation π_c, find the best permutation π of V that agrees with π_c. We say that π and π_c *agree* if for all $c_i, c_j \in C$ we have that $c_i <_\pi c_j$ if and only of $c_i <_{\pi_c} c_j$. In other words, the relative ordering π imposes on C is precisely π_c. Thus, at a cost of a factor of $k!$ in the running time we can assume that there exists an optimal permutation π such that $c_1 <_\pi c_2 <_\pi \ldots <_\pi c_k$.

Definition 2.8.3 *Let π_c be an ordering of C such that $c_1 <_{\pi_c} c_2 <_{\pi_c} \ldots <_{\pi_c} c_k$. We define C_i to be $\{c_1, c_2, \ldots, c_i\}$ for every i such that $1 \le i \le k$.*

Let I be the independent set $V \setminus C$. We associate a *type* with each vertex in I. A type is simply a subset of C.

Definition 2.8.4 *Let I be the independent set $V \setminus C$. The* type *of a vertex v in I is $N(v)$. For a type $S \subseteq C$ the set $I(S)$ is the set of all vertices in I of type S.*

Notice that two vertices of the same type are indistinguishable up to automorphisms of G, and that there are 2^k different types.

Observe that every vertex of I is either mapped between two vertices of C, to the left of c_1 or to the right of c_k by a permutation π. For a permutation π we say that a vertex v is at *location* 0 if $v <_\pi c_1$ and at location i if i is the largest integer such that $c_i <_\pi v$. The set of vertices that are at location i is denoted by L_i. We define the *inner order* of π at location i to be the permutation defined by π restricted to L_i.

The task of finding an optimal permutation can be divided into two parts. The first part is to partition the set I into $L_0, \ldots, L_k$, while the second part consists of finding an optimal inner order at all locations. One should notice that partitioning I into $L_0, \ldots, L_k$ amounts to deciding *how many* vertices of each type are at location i for each i. Observe that permuting the inner order of π at location i does not change the imbalance of any single vertex, where the imbalance of a vertex v is $|L_\pi(v) - R_\pi(v)|$. Hence, the inner orders are in fact irrelevant and finding the optimal ordering of the vertices thus reduces to finding the right partition of I into $L_0, \ldots, L_k$. We formalize this as an instance of p-OPT-ILP.

For a type S and location i we let x_S^i be a variable that encodes the number of vertices of type S that are at location i. Also, for every vertex c_i in C we have a variable y_i that represents the imbalance of c_i. In order to represent a feasible permutation, all the variables must be non-negative. Also the variables x_S^i must satisfy that for every type S, $\sum_{i=0}^{k} x_S^i = |I(S)|$. For every vertex c_i of the vertex cover let $e_i = |N(c_i) \cap C_{i-1}| - |N(c_i) \cap (C \setminus C_i)|$ be a constant. Finally for every $c_i \in C$ we add the constraint

$$y_i = \Big|e_i + \sum_{\{S \subseteq C | c_i \in S\}} \big(\sum_{j=0}^{i-1} x_S^j - \sum_{j=i}^{k} x_S^j\big)\Big|.$$

One should notice that the last set of constraints is not a set of linear constraints. However, we can guess the sign of

$$y_i' = e_i + \sum_{\{S \subseteq C | c_i \in S\}} \big(\sum_{j=0}^{i-1} x_S^j - \sum_{j=i}^{k} x_S^j\big)$$

for every i in an optimal solution. This increases the running time by a factor of 2^k. For every i we let t_i take the value 1 if we have guessed that $y_i' \geq 0$ and we let t_i take the value -1 if we have guessed that $y_i' < 0$. We can now replace the non-linear constraints with the linear constraints $y_i = t_i y_i'$ for every i. Finally, for every type S and location i, let z_S^i be the constant $\big||S \cap C_i| - |S \cap (C \setminus C_i)|\big|$. We are now ready to formulate the integer linear program.

$$\begin{array}{lll}
\text{min} & \sum_{i=1}^{k} t_i \cdot y_i + \sum_{S \subseteq C} z_S^i \cdot x_S^i & \\
\text{such that} & \sum_{i} x_S^i = |I(S)| & \text{for all } i \in \{0, \ldots, k\}, S \subseteq C \\
& y_i = t_i e_i + \sum_{\{S \subseteq C | c_i \in S\}} \big(\sum_{j=0}^{i-1} t_i x_S^j - \sum_{j=i}^{k} t_i x_S^j\big) & \text{for all } i \in \{1, \ldots, k\} \\
& x_S^i,\ y_i \geq 0 & \text{for all } i \in \{0, \ldots, k\}, S \subseteq C
\end{array}$$

Since the value of $f_{im}(\pi)$ is bounded by n^2 and the value of any variable in the integer linear program is bounded by n, Theorem 2.8.2 implies that this integer linear program can be solved in FPT time, thus implying the following theorem.

Theorem 2.8.5 *The* IMBALANCE *problem parameterized by the vertex cover number of the input graph is fixed parameter tractable.*

The proof of Theorem 2.8.5 serves as an example for how to apply Theorem 2.8.2 to give FPT algorithms. The result of Lenstra was extended by Khachiyan and Porkolab [92] to semidefinite integer programming. In their work, they show that if Y is a convex set in R^k defined by polynomial inequalities and equations of degree at most $d \geq 2$, with integer coefficients of binary length at most l, then for fixed k, the problem of computing an optimal integral solution y^* to the problem min $\{y_k \mid y(y_1, \ldots, y_l) \in Y \cup \mathrm{Z}^k\}$ admits an FPT algorithm. Their algorithm was further improved by Heinz [84] in the specific case of minimizing a polynomial $\hat{F}$ on the set of integer points described by an inequality system $F_i \leq 0$, $1 \leq i \leq s$ where the F_i's are quasiconvex polynomials in p variables with integer coefficients. It would be interesting to see if these even more general results can be useful for showing problems fixed parameter tractable.

Chapter 3

Running Time Lower Bounds

In this chapter we discuss techniques for showing lower bounds for the running time of algorithms for parameterized problems, up to certain complexity theoretic assumptions. We take an "engineering" approach to the topic, that is, we do not dwell on the underlying complexity theory. Instead we discuss the types of bounds one can obtain under different assumptions and give examples of how to show such bounds.

3.1 NP-hardness vs. ParaNP-hardness

NP-hardness of a parameterized problem rules out algorithms with running time $n^{O(1)}$, assuming P $\neq$ NP . On the other hand, one might have an algorithm with running time $n^{f(k)}$. For instance, the INDEPENDENT SET problem admits a simple $O(n^k)$ time algorithm: try all possible vertex subsets of size k and check whether they form an independent set. However, not all problems have algorithms with running time of this form. Some parameterized problems, as we demonstrate below, remain NP-hard even when k is a fixed integer. Parameterized problems that are NP-hard for every fixed value of k above some threshold K are called ParaNP-hard. For these problems one can not hope for algorithms with running time on the form $O(n^{O(f(k))})$, let alone FPT algorithms.

Theorem 3.1.1 ([126]) GRAPH COLORING *is* ParaNP-*hard*

Proof. We show that for every fixed $k \geq 3$, the problem of deciding whether an input graph G is k-colorable is NP-complete. A reduction showing that GRAPH COLORING is NP-complete for $k = 3$ is given in for example the book of Sipser [126]. For a given graph G we can make a graph G' by adding a new vertex adjacent to all vertices of G. It follows that G is k-colorable if and only if G' is $k+1$-colorable and hence GRAPH COLORING is NP-complete for every fixed $k > 3$. ∎

Quite a few problems share this property, and for many of them showing ParaNP-hardness turned out to be easier than showing $W[1]$-hardness (explained below). Examples include TREELENGTH [103], METRIC EMBEDDING INTO THE LINE [60] and MINIMUM FILL-IN parameterized by the number of vertices that has to be deleted from the input graph to make it chordal [106].

3.2 Hardness for W[1]

How can we show that a problem does not have an algorithm with running time $f(k) \cdot n^{O(1)}$? One approach would be to show NP-hardness. But with this method, we can not distinguish between problems that are solvable in time $n^{f(k)}$ from problems solvable in time $f(k) \cdot n^{O(1)}$. To be able to do this, Downey and Fellows [52] introduced the W-hierarchy. The hierarchy consists of a complexity class W[t] for every integer $t \geq 1$ such that W[t] $\subset$ W[$t+1$] for all t. Downey and Fellows [52] proved that FPT $\subseteq$ W[1] $\subseteq$ W[2] $\ldots \subseteq$ W[t] and conjectured that strict containment holds.

In particular, the assumption FPT $\subset$ W[1] is the fundamental complexity theoretic assumption in Parameterized Complexity. The reason for this is that the assumption is a natural parameterized analogue of the conjecture that P $\neq$ NP. The assumption P $\neq$ NP can be reformulated as "The NON-DETERMINISTIC TURNING MACHINE ACCEPTANCE problem can not be solvable in polynomial time". In this problem we are given a non-deterministic turing machine M, a string s and an integer k coded in unary. The question is whether M can make its non-deterministic choices in such a way that it accepts s in at most k steps. The intuition behind the P $\neq$ NP conjecture is that this problem is so general and random that it is not likely to be in P. Similarly, one would not expect the problem parameterized by k to be fixed parameter tractable.

From our "engineering" viewpoint, how can we use the assumption that FPT $\neq$ W[1] to rule out $f(k) \cdot n^{O(1)}$ time algorithms for a particular problem? Downey and Fellows showed that the INDEPENDENT SET problem is not in FPT unless FPT = W[1]. If we can show for a particular parameterized problem Π, that if Π is fixed parameter tractable then so is INDEPENDENT SET, then this implies that $\Pi \notin$ FPT unless FPT = W[1]. To this end, we define the notion of FPT-*reductions.*

Definition 3.2.1 ([52]) *Let P and Q be parameterized problems. We say that P is* FPT*-reducible to Q, written $P \leq_{FPT} Q$, if there exists an algorithm that given as input an instance (x, k), runs in time $|x|^{O(1)} f(k)$ for some function $f : \mathbb{N} :\rightarrow \mathbb{N}$, and outputs an instance (x', k') such that (a) $(x, k) \in P$ if and only $(x', k') = f(x, k) \in Q$ and (b) $k' \leq g(k)$ for some function $g : \mathbb{N} :\rightarrow \mathbb{N}$.*

The class W[1] is the set of all parameterized problem that are FPT-reducible to the parameterized NON-DETERMINISTIC TURNING MACHINE ACCEPTANCE problem. A parameterized problem is said to be W[1]-*hard* if all problems in W[1] FPT-reduce to it. The following theorem follows directly from Definition 3.2.1.

Theorem 3.2.2 ([52]) *Let P and Q be parameterized problems. If $P \leq_{FPT} Q$ and Q is in* FPT *then P is in* FPT*. Furthermore, if $P \leq_{FPT} Q$ and P is* W[1]-hard *then Q is* W[1]-hard*.*

Theorem 3.2.2 implies that a W[1]-*hard* problem can not be in FPT unless FPT = W[1]. As mentioned above, it was proved in [52] that Independent Set is W[1]-*hard.* We now give two examples of FPT-reductions. First, we show that the Multicolor Clique problem is W[1]-*hard.*

> Multicolor Clique: Given an integer k and a connected undirected graph $G = (V[1] \cup V[2] \cdots \cup C[k], E)$ such that for every i the vertices of $V[i]$ induce an independent set, is there a k-clique C in G?

The approach for using the Multicolor Clique problem in reductions is described in [62], and has been proven to be very useful in showing hardness results in Parameterized Complexity.

Theorem 3.2.3 Multicolor Clique *is* W[1]-*hard.*

Proof. We reduce from the Independent Set problem. Given an instance (G, k) to Independent Set we construct a new graph $G' = (V', E')$ as follows. For each vertex $v \in V(G)$ we make k copies of v in V' with the i'th copy being colored with the i'th color. For every pair u,$v \in V(G)$ such that $uv \notin E(G)$ we add edges between all copies of u and all copies of v with different colors. It is easy to see that G has an independent set of size k if and only if G' contains a clique of size k. This concludes the proof. ∎

One should notice that the reduction produces instances to Multicolor Clique with a quite specific structure. In particular, all color classes have the same size and the number of edges between every pair of color classes is the same. It is often helpful to exploit this fact when reducing from Multicolor Clique to a specific problem. We now give an example of a slightly more involved FPT-reduction.

Theorem 3.2.4 Dominating Set *is* W[1]-*hard.*

Proof. We reduce from the Multicolor Clique problem. Given an instance (G, k) to Multicolor Clique we construct a new graph G'. For every $i \leq k$ let V_i be the set of vertices in G colored i and for every pair of distinct integers $i, j \leq k$ let $E_{i,j}$ be the set of edges in $G[V_i \cup V_j]$. We start making G' by taking a copy of V_i for every $i \leq k$ and making this copy into a clique. Now, for every

$i \leq k$ we add a set S_i of $k+1$ vertices and make them adjacent to all vertices of V_i. Finally, for every pair of distinct integers $i, j \leq k$ we consider the edges in $E_{i,j}$.For every pair of vertices $u \in V_i$ and $v \in V_j$ such that $uv \notin E_{i,j}$ we add a vertex x_{uv} and make it adjacent to all vertices in $V_i \setminus \{u\}$ and all vertices in $V_j \setminus \{v\}$. This concludes the construction. We argue that G contains a k-clique if and only if G' has a dominating set of size at most k.

If G contains a k-clique C then C is a dominating set of G'. In the other direction, suppose G' has a dominating set S of size at most k. If for some i, $S \cap V_i = \emptyset$ then $S_i \subseteq S$, contradicting that S has size at most k. Hence for every $i \leq k$, $S \cap V_i \neq \emptyset$ and thus S contains exactly one vertex v_i from V_i for each i, and S contains no other vertices. Finally, we argue that S is a clique in G. Suppose that $v_i v_j \notin E_{i,j}$. Then there is a vertex x in $V(G')$ with neighbourhood $V_i \setminus \{u\}$ and $V_j \setminus \{v\}$. This x is not in S and has no neighbours in S contradicting that S is a dominating set of G'. ∎

In fact, it was an FPT-reduction from Independent Set to Dominating Set that was the startpoint of the complexity part of parameterized algorithms and complexity. In 1989, Fellows realised that one could give a FPT-reduction from Independent Set to Dominating Set, but that it did not seem plausible to give a reduction in the other direction. Later, Downey and Fellows proved that the Dominating Set problem in fact is complete for the class W[2] while Independent Set is complete for W[1] [52]. It is therefore unlikely that an FPT-reduction from Dominating Set to Independent Set can exist. The theorem 3.2.4 is due to Downey and Fellows. The proof presented in this thesis is somewhat simpler than the original proof and due to the author. For a thorough introduction to the W-hierarchy we refer the reader to the books of Downey and Fellows [52] and Flum and Grohe [66].

3.3 Hardness for W[2]

Even though it was shown by Downey and Fellows that Dominating Set is complete for W[2] while Independent Set is complete for W[1] [52], there are problems for which it is easier to give FPT-reductions from Dominating Set than from Independent Set. A typical example is the Steiner Tree problem, parameterized by the number of *non-terminals* in the solution. In the Steiner Tree problem we are given a connected graph G together with a subset X of $V(G)$ and an integer k. The vertices in the X are called *terminals*. The objective is to find a subtree T of G containing all terminals and at most k non-terminals. Such a tree is called a *steiner tree* of G. Here we give a proof that the Steiner Tree problem parameterized by the number of non-terminals in the steiner tree is W[2] hard. The proof of the following theorem first appeared in [17].

Theorem 3.3.1 ([17]) *The* Steiner Tree *problem parameterized by* $|V(T) \setminus$

$X|$ *is* W[2]*-hard.*

Proof. We reduce from the Dominating Set problem. For an instance (G, k) we build a graph G' as follows. We make two copies of $V(G)$, call them X' and N. The copy $x \in X'$ of each vertex $v \in V(G)$ is made adjacent to the copies in N of the vertices in $N(v)$. Finally the vertex set X is obtained from X' by adding a single vertex u_X and making it adjacent to all vertices in N. This concludes the construction of G'. We prove that G has a dominating set of size at most k if and only if G' has a steiner tree with at most $|X| + k$ vertices.

In one direction, suppose G has a dominating set S on k vertices. Let S' be the copy of S in N. Then the graph $G'[X \cup S']$ is connected. Let T be a spanning tree of $G'[X \cup S']$, then T is a steiner tree of G' with at most $|X| + k$ vertices. In the other direction, suppose G' has a steiner tree T on at most $|X| + k$ vertices and let $S' = V(T) \cap N$. Then $|S'| \leq k$ and since X is an independent set in G' every vertex in X has a neighbour in S'. Thus, if we let S be the copy of S' in $V(G)$ then S is a dominating set of G of size at most k. ∎

Chapter 4

Kernelization

Polynomial time preprocessing to reduce instance size is one of the most widely used approaches to tackle computationally hard problems. A natural question in this regard is how to measure the quality of preprocessing rules proposed for a specific problem. Parameterized complexity provides a natural mathematical framework to give performance guarantees of preprocessing rules. A parameterized problem is said to admit an $f(k)$ *kernel* if there is a polynomial time algorithm, called a *kernelization* algorithm, that reduces the input instance down to an instance with size bounded by $f(k)$ in k, while preserving the answer. This reduced instance is called an $f(k)$ *kernel* for the problem.

Definition 4.0.2 *A* kernelization algorithm*, or in short, a kernel for a parameterized problem $Q \subseteq \Sigma^* \times \mathbb{N}$ is an algorithm that given $(x, k) \in \Sigma^* \times \mathbb{N}$ outputs in time polynomial in $|x| + k$ a pair $(x', k') \in \Sigma^* \times \mathbb{N}$ such that (a) $(x, k) \in Q$ if and only if $(x', k') \in Q$ and (b) $|x'|, k \leq g(k)$, where g is an arbitrary computable function. The function g is referred to as the size of the kernel. If g is a polynomial function then we say that Q admits a polynomial kernel.*

It is easy to see that if a decidable problem admits an $f(k)$ kernel for some function f, then the problem is FPT. Interestingly, the converse also holds, that is, if a problem is FPT then it admits an $f(k)$ kernel for some function f [116]. The proof of this fact is quite simple, and we present it here.

Fact 4.0.3 (Folklore, [116]) *If a parameterized problem Π is* FPT *then Π admits an $f(k)$ kernel for some function f.*

Proof. Suppose there is a decision algorithm for Π running in $f(k)n^c$ time for some function f and constant c. Given an instance (I, k) with $|I| = n$, if $n \geq f(k)$ then we run the decision algorithm on the instance in time $f(k)n^c \leq n^{c+1}$. If the decision algorithm outputs *yes*, the kernelization algorithm outputs a constant size *yes* instance, and if the decision algorithm outputs *no*, the kernelization algorithm outputs a constant size *no* instance. On the other hand, if $n < f(k)$ the kernelization algorithm just outputs (I, k). This yields an $f(k)$ kernel for the problem. ∎

Fact 4.0.3 implies that a problem has a kernel if and only if it is fixed parameter tractable. However, we are interested in kernels that are as small as possible, and a kernel obtained using Fact 4.0.3 has size that equals the dependence on k in the running time of the best known FPT algorithm for the problem. The question is - can we do better? The answer is that quite often we can. In fact, for many problems we can obtain k^c kernels, called *polynomial* kernels. In this chapter we suvey some of the known techniques for showing that problems admit polynomial kernels.

4.1 Reduction Rules

We have already seen the main tool for showing that a problem admits an $f(k)$-kernel in a different context. Namely, we briefly touched upon *reduction rules* in Section 2.2. Recall that a reduction rule is a polynomial time algorithm that transforms an instance (I, k) of a parameterized problem to an 'equivalent and simpler instance (I', k'), where equivalent means that (I, k) is a *yes* instance if and only if (I', k') is. In Section 2.2 we used reduction rules to identify "good" structures to branch on. Here we will show that for some problems we can give reduction rules such that any instance to which no reduction rules can be applied must have size bounded by a function of k. This will immediately give us the desired kernel.

We show how a simple set of reduction rules gives a k^2 kernel for the Point Line Cover problem. An instance of this problem is a pair (S, k) where S is a set of n points in the plane with no two points in the exact same spot and k is an integer. The objective is to find a set C of k lines in the plane such that every point lies on some line in C. For a line l in the plane let $P(l)$ be the set of points that are covered by l. To tackle the problem, it is enough to make the following observations:

1. It is sufficient to only consider lines containing at least 2 points.

2. Any two lines intersect in at most one point.

Hence any line that contains at least $k+1$ points must be in the solution set S, since otherwise S must contain at least $k+1$ lines. This yields the following reduction rule.

RedRule 4.1.0.1 *If some line l covers at least $k+1$ points, return $(S \setminus P(l), k-1)$.*

Rule 4.1.0.1 is correct since any line except for l covers at most one point in $P(l)$. Hence if the solution set C does not contain l then it must contain at least $P(l) > k$ lines.

Theorem 4.1.1 *The* Point Line Cover *problem admits a k^2 kernel.*

Proof. Consider an instance (S, k) to which Rule 4.1.0.1 can not be applied. We argue that if $|S| > k^2$, then (S, k) is a no-instance. Suppose for contradiction that there is a set C of at most k lines such that the lines in C together cover all points in S. Since Rule 4.1.0.1 can not be applied every line in C covers at most k points in S. Thus the lines in C cover at most k^2 points of S in total, contradicting that S is covered. ∎

Theorem 4.1.1 imediately yields a $O(k^{4k}n^{O(1)})$ time algorithm for the Point Line Cover problem: after obtaining the k^2 kernel, iterate over all $\binom{\binom{k^2}{2}}{k}$ ways to chose k lines and check whether they cover all the points. In fact, a $O(k^2)$ kernel and a $O(k^{O(k)}n^{O(1)})$ time algorithm for this problem is a corollary of a result by Langerman and Morin [100]. However, no $o(k^2)$ sized kernel or $O(2^{o(k^2)}n^{O(1)})$ time algorithm is known for the problem to this date and remains an interesting open problem.

4.2 Crown Reductions

A *crown decomposition* of a graph G is a decomposition of $V(G)$ into $C \uplus H \uplus R$ such that $N(C) \subseteq H$, $G[C]$ is an independent set and there is a matching from H to C. The set C is called the *crown*, H is called the *head* and R is called the *rest* or *body* of the decomposition. For many problems, if a crown decomposition of the input graph, or a graph modelling the problem, is found then the crown can be removed or reduced. We give an example by giving a kernel for the Maximum Satisfiability problem. In the Maximum Satisfiability problem we are given a formula ϕ in conjunctive normal form and an integer k. The formula has n variables and m clauses and the question is whether there is an assignment to the variables that satisfies at least k of the clauses. We give a kernel based on crown reductions with less than k variables and less than $2k$ clauses. Let G_ϕ be the *variable-clause incidence graph* of ϕ. That is, G_ϕ is a bipartite graph with one bipartition X corresponding to the variables of ϕ and a bipartition Y corresponding to the clauses. For a vertex $x \in X$ we will refer to x as both the vertex in G_ϕ and the corresponding variable in ϕ. Similarly, for a vertex $c \in Y$ we will refer to c as both the vertex in G_ϕ and the corresponding clause in ϕ. In G_ϕ there is an edge between a variable $x \in X$ and a clause $c \in Y$ if x or its negation occurs in c. We assume that every clause of ϕ contains at least one variable.

Theorem 4.2.1 *The* Maximum Satisfiability *problem admits a kernel with less than k variables and less than $2k$ clauses.*

Proof. If we assign values to the variables uniformly at random, linearity of expectation yields that the expected number of satisfied clauses it at least $m/2$. Since there has to be at least one assigment satisfying at least the expected number of clauses this means that if $m \geq 2k$ then (ϕ, k) is a yes-instance. In the rest of this section we show how to give a kernel where $n < k$. Whenever possible we apply a cleaning rule; if some variable does not occur in any clauses, remove the variable.

Let G_ϕ be the variable-clause incidence graph of ϕ, X and Y be the bipartitions of G_ϕ corresponding to variables and clauses respectively. If there is a matching from X to Y in G_ϕ there is an assignment to the variables satisfying at least n clauses. This is true because we can set each variable in X in such a way that the clause matched to it becomes satisfied. In this manner, at least $|X|$ clauses are satisfied. We now show that if ϕ has at least k variables then we can, in polynomial time, either reduce ϕ to an equivalent smaller instance or find an assignment to the variables satisfying at least k clauses.

Suppose ϕ has at least k variables. Using Hall's Theorem and an algorithm for Maximum Matching we can in polynomial time find either a matching from X to Y or an inclusion minimal set $C \subseteq X$ such that $|N(C)| < |C|$. If we found a matching we are done, as we can satisfy at least $|X| \geq k$ clauses. So suppose we found a set C as described. Let H be $N(C)$ and $R = V(G_\phi) \setminus (C \cup H)$. Clearly, $N(C) \subseteq H$, $N(R) \subseteq H$ and $G[C]$ is an independent set. Furthermore, for a vertex $x \in C$ we have that there is a matching from $C \setminus x$ to H since $|N(C')| \geq |C'|$ for any $C' \subseteq (C \setminus x)$. Since $|C > H|$ the matching from $C \setminus x$ to H is in fact a matching from H to C. Hence $C \uplus H \uplus R$ is a crown decomposition of G_ϕ.

We prove that all clauses in H are satisfied in every truth assignment to the variables satisfying the maximum number of clauses. Indeed, consider any assignment ψ of values to the variables that does not satisfy all clauses in H. For every variable y in $C \setminus \{x\}$ change the value of y such that the clause in H matched to y is satisfied. Let ψ' be the new assignment obtained from ψ in this manner. Since $N(C) \subseteq H$ and ψ' satisfies all clauses in H, ψ' satisfies more clauses than ψ does, and hence ψ can not be an assigment satisfying the maximum number of clauses.

The argument above shows that (ϕ, k) is a yes instance to Maximum Satisfiability if and only if $(\phi \setminus (C \cup H), k - |H|)$ is. This gives rise to a simple reduction rule: remove $(C \cup H)$ from ϕ and decrease k by $|H|$. This completes the proof of the theorem. ∎

To the author's best knowledge, the best kernel known before this result is a kernel with $2k$ variables and $O(k^2)$ clauses [116].

4.3 LP based kernelization

One of the classical results in algorithms is a polynomial time algorithm to solve LINEAR PROGRAMMING [93]. This result has proved extremely useful in the design of approximation algorithms. In particular, any problem in NP can be formulated as an *integer linear program* (ILP), an LP with the additional restriction that the variables only are allowed to take integer values. Since INTEGER LINEAR PROGRAMMING is NP-complete we can not hope to solve the integer linear program in polynomial time. Instead we settle for an approximate solution by solving the corresponding *linear programming relaxation* of the ILP, which is just the ILP without the integrality constraint on the variables. A (not necessarily optimal) solution to the ILP is obtained by rounding the variables in an optimal solution to the LP relaxation in an appropriate way. LINEAR PROGRAMMING is also helpful to give problem kernels. We show how to obtain a $2k$ kernel for the VERTEX COVER problem by applying Linear Programming. This kernel is due to [29, 115].

Given a graph G and integer k we construct an ILP with n variables, one variable x_v for each vertex $v \in V(G)$. Setting the variable x_v to 1 means that v goes into the vertex cover, while setting x_v to 0 means that v is not in the vertex cover. This yields the following ILP:

$$\begin{aligned} \min \sum_{v \in V(G)} x_v & \qquad \text{subject to:} \\ x_u + x_v \geq 1 & \quad \text{for every } uv \in E(G) \\ x_v \in \{0,1\} & \quad \text{for every } v \in V(G) \end{aligned}$$

Clearly the optimal value of the ILP is at most k if and only if G has a vertex cover of size at most k. We relax the ILP by replacing the constraint $x_v \in \{0,1\}$ for every $v \in V(G)$ with the constraint $0 \leq x_v \leq 1$ for every $v \in V(G)$. Now we solve this LP in polynomial time. If the optimal value of the LP is greater then k then it must be greater than k for the ILP as well and we return that G has no vertex cover of size at most k. Otherwise, let x_v^* be the value of x_v in the produced optimal solution to the LP. We partition the vertex set of G into three sets, $C = \{v : x_v^* < 1/2\}$, $H = \{v : x_v^* > 1/2\}$ and $R = \{v : x_v^* = 1/2\}$. The name choice for the three sets is not arbitrary, as $V(G) = C \uplus H \uplus R$ is in fact a crown decomposition of G. No edge can go between two vertices of C or between a vertex of C and a vertex of R since this would violate the constraint that $x_u + x_v \geq 1$ for every $uv \in E(G)$. Furthermore, if for any subset $H' \subseteq H$ we have $|H'| > |N(H) \cap C|$ then decreasing the value of x_v by ϵ for every $v \in H'$ and increasing the value of x_v^* by ϵ for every $v \in N(H) \cap C$ would yield a feasible solution to the LP with a lower value of $\min \sum_{v \in V(G)} x_v$, contradictiong that x^*

was an optimal solution to the LP. By Halls Theorem H is matched into C in G. For a vertex $v \in H$, let $m(v)$ be the vertex in C matched to v.

Theorem 4.3.1 *The* VERTEX COVER *problem admits a* $2k$ *kernel.*

Proof. We first prove that there is an optimal vertex cover S of G containing all vertices of H and no vertices of C. Let S' be an optimal vertex cover of G. For every vertex $v \in H \setminus S'$, $m(v)$ must be in S'. Hence $S = (S' \cup H) \setminus C$ is a vertex cover of G with $|S| \leq |S'|$. This gives rise to a reduction rule since (G, k) is a yes instance to VERTEX COVER if and only if $(G \setminus (C \cup S), k - |H|)$ is. If H and C are non-empty, remove H and C from the graph and reduce k by $|H|$. Now, if C is empty then $x^*v \geq 1/2$ for all $v \in V(G)$. But $\sum_{v \in V(G)} x_v \leq k$ so $|V(G)| \leq 2k$.
∎

Chapter 5

Kernelization Lower Bounds

In Chapter 4 we saw that a parameterized problem admits a kernel if and only if it is FPT, and that for many fixed parameter tractable problems we are able to obtain kernels of size polynomial in the parameter k. In Chapter 3 we discussed methods for showing that a problem is not FPT under different complexity-theoretic assumptions. Of course, due to Fact 4.0.3 if we can show that a problem is not FPT under a certain complexity-theoretic assumption then under the same assumption the problem does not admit an $f(k)$ kernel for *any* function f. However, this method will not let us separate "good", that is polynomial, kernels from "bad" $f(k)$ kernels. It is only very recently that a methodology to rule out polynomial kernels has been developed [20, 70]. In this chapter we survey the tecniques that have been developed to show kernelization lower bounds.

5.1 Composition Algorithms

Consider the Longest Path problem. In Section 2.7 we saw an FPT algorithm for this problem. Is it feasible that it also admits a polynomial kernel? We argue that intuitively this should not be possible. Consider a large set $(G_1, k), (G_2, k), \ldots, (G_t, k)$ of instances to the Longest Path problem. If we make a new graph G by just taking the disjoint union of the graphs $G_1 \ldots G_t$ we see that G contains a path of length k if and only if G_i contains a path of length k for some $i \leq t$. Suppose the Longest Path problem had a polynomial kernel, and we ran the kernelization algorithm on G. Then this algorithm would in polynomial time return a new instance (G', k') such that $|V(G)| = k^{O(1)}$, a number potentially much smaller than t. This means that in some sense, the kernelization algorithm considers the instances $(G_1, k), (G_2, k) \ldots (G_t, k)$ and in *polynomial time* figures out which of the instances are the most likely to contain a path of length k. However, at least intuitively, this seems almost as difficult as solving the instances themselves and since the Longest Path problem is NP-complete, this seems unlikely. We now formalize this intuition.

Definition 5.1.1 [Distillation [20]]

- *An OR-distillation algorithm for a language $L \subseteq \Sigma^*$ is an algorithm that receives as input a sequence $x_1, \ldots, x_t$, with $x_i \in \Sigma^*$ for each $1 \leq i \leq t$, uses time polynomial in $\sum_{i=1}^{t} |x_i|$, and outputs $y \in \Sigma^*$ with (a) $y \in L \iff x_i \in L$ for* some *$1 \leq i \leq t$ and (b) $|y|$ is polynomial in $\max_{i \leq t} |x_i|$. A language L is OR-distillable if there is a OR-distillation algorithm for it.*

- *An AND-distillation algorithm for a language $L \subseteq \Sigma^*$ is an algorithm that receives as input a sequence $x_1, \ldots, x_t$, with $x_i \in \Sigma^*$ for each $1 \leq i \leq t$, uses time polynomial in $\sum_{i=1}^{t} |x_i|$, and outputs $y \in \Sigma^*$ with (a) $y \in L \iff x_i \in L$ for* all *$1 \leq i \leq t$ and (b) $|y|$ is polynomial in $\max_{i \leq t} |x_i|$. A language L is AND-distillable if there is an AND-distillation algorithm for it.*

Observe that the notion of distillation is defined for unparameterized problems. Bodlaender et al. [20] conjectured that no NP-complete language can have an OR-distillation or an AND-distillation algorithm.

Conjecture 5.1.2 (OR-Distillation Conjecture [20]) *No* NP*-complete language L is OR-distillable.*

Conjecture 5.1.3 (AND-Distillation Conjecture [20]) *No* NP*-complete language L is AND-distillable.*

One should notice that if any NP-complete language is distillable, then so are all of them. Fortnow and Santhanam [70] were able to connect the OR-Distillation Conjecture to a well-known conjecture in classical complexity. In particular they proved that if the OR-Distillation Conjecture fails, the *polynomial time hierarchy* [131] collapses to the third level, a collapse that is deemed unlikely. No such connection is currently known for the AND-Distillation Conjecture, and for reasons soon to become apparent, a proof of such a connection would have significant impact in Parameterized Complexity. By PH=Σ_p^3 we will denote the complexity-theoretic event that the polynomial time hierarchy collapses to the third level.

Theorem 5.1.4 ([70]) *If the OR-Distillation Conjecture fails, then* PH=Σ_p^3.

We are now ready to define the parameterized analogue of distillation algorithms and connect this notion to the Conjectures 5.1.2 and 5.1.3

Definition 5.1.5 [Composition [20]]

- *A composition algorithm (also called OR-composition algorithm) for a parameterized problem $\Pi \subseteq \Sigma^* \times \mathbb{N}$ is an algorithm that receives as input a sequence $((x_1, k), \ldots, (x_t, k))$, with $(x_i, k) \in \Sigma^* \times \mathbb{N}^+$ for each $1 \leq i \leq t$, uses time polynomial in $\sum_{i=1}^{t} |x_i| + k$, and outputs $(y, k') \in \Sigma^* \times \mathbb{N}^+$ with (a) $(y, k') \in \Pi \iff (x_i, k) \in \Pi$ for* some *$1 \leq i \leq t$ and (b) k' is polynomial in k. A parameterized problem is compositional (or* OR-compositional*) if there is a composition algorithm for it.*

- *An AND-composition algorithm for a parameterized problem $\Pi \subseteq \Sigma^* \times \mathbb{N}$ is an algorithm that receives as input a sequence $((x_1, k), \ldots, (x_t, k))$, with $(x_i, k) \in \Sigma^* \times \mathbb{N}^+$ for each $1 \leq i \leq t$, uses time polynomial in $\sum_{i=1}^{t} |x_i| + k$, and outputs $(y, k') \in \Sigma^* \times \mathbb{N}^+$ with (a) $(y, k') \in \Pi \iff (x_i, k) \in \Pi$ for* all *$1 \leq i \leq t$ and (b) k' is polynomial in k. A parameterized problem is AND-compositional if there is an AND-composition algorithm for it.*

Composition and distillation algorithms are very similar. The main difference between the two notions is that the restriction on output size for distillation algorithms is replaced by a restriction on the parameter size for the instance the composition algorithm outputs. We define the notion of the *unparameterized version* of a parameterized problem L. The mapping of parameterized problems to unparameterized problems is done by mapping (x, k) to the string $x\#1^k$, where $\# \notin \Sigma$ denotes the blank letter and 1 is an arbitrary letter in Σ. In this way, the unparameterized version of a parameterized problem Π is the language $\widetilde{\Pi} = \{x\#1^k \mid (x, k) \in \Pi\}$. The following theorem yields the desired connection between the two notions.

Theorem 5.1.6 ([20, 70]) *Let Π be a compositional parameterized problem whose unparameterized version $\widetilde{\Pi}$ is* NP*-complete. Then, if Π has a polynomial kernel then* PH=Σ_p^3. *Similarly, let Π be an AND-compositional parameterized problem whose unparameterized version $\widetilde{\Pi}$ is* NP*-complete. Then, if Π has a polynomial kernel the AND-Distillation Conjecture fails.*

We can now formalize the discussion from the beginning of this section.

Theorem 5.1.7 ([20]) Longest Path *does not admit a polynomial kernel unless* PH=Σ_p^3.

Proof. The unparameterized version of Longest Path is known to be NP-complete [72]. We now give a composition algorithm for the problem. Given a sequence $(G_1, k) \ldots (G_t, k)$ of instances we output (G, k) where G is the disjoint union of $G_1 \ldots G_t$. Clearly G contains a path of length k if and only if G_i contains a path of length k for some $i \leq t$. By Theorem 5.1.6 Longest Path does not have a polynomial kernel unless PH=Σ_p^3. ∎

An identical proof can be used to show that the Longest Cycle problem does not admit a polynomial kernel unless PH=Σ_p^3. For many problems, it is easy to give AND-composition algorithms. For instance, the "disjoint union" trick yields AND-composition algorithms for the Treewidth, Cutwidth and Pathwidth problems, among many others. Coupled with Theorem 5.1.6 this implies that these problems do not admit polynomial kernels unless the AND-Distillation Conjecture fails. However, to this date, there is no strong complexity theoretic evidence known to support the AND-Distillation Conjecture. Therefore it would be interesting to see if such evidence could be provided.

5.2 Polynomial Parameter Transformations

For some problems, obtaining a composition algorithm directly is a difficult task. Instead, we can give a reduction from a problem that provably has no polynomial kernel unless PH=Σ_p^3 to the problem in question such that a polynomial kernel for the problem considered would give a kernel for the problem we reduced from. We now define the notion of *polynomial parameter transformations.*

Definition 5.2.1 ([24]) *Let P and Q be parameterized problems. We say that P is polynomial parameter reducible to Q, written $P \leq_{Ptp} Q$, if there exists a polynomial time computable function $f : \Sigma^* \times \mathbb{N} \rightarrow \Sigma^* \times \mathbb{N}$ and a polynomial p, such that for all $(x, k) \in \Sigma^* \times \mathbb{N}$ (a) $(x, k) \in P$ if and only $(x', k') = f(x, k) \in Q$ and (b) $k' \leq p(k)$. The function f is called* polynomial parameter transformation.

Proposition 5.2.2 ([24]) *Let P and Q be the parameterized problems and $\tilde{P}$ and $\tilde{Q}$ be the unparameterized versions of P and Q respectively. Suppose that $\tilde{P}$ is* NP*-complete and $\tilde{Q}$ is in* NP*. Furthermore if there is a polynomial parameter transformation from P to Q, then if Q has a polynomial kernel then P also has a polynomial kernel.*

Proposition 5.2.2 shows how to use polynomial parameter transformations to show kernelization lower bounds. A notion similar to polynomial parameter transformation was independently used by Fernau et al. [65] albeit without being explicitly defined. We now give an example of how Proposition 5.2.2 can be useful for showing that a problem does not admit a polynomial kernel. In particular, we show that the Path Packing problem does not admit a polynomial kernel unless PH=Σ_p^3. In this problem you are given a graph G together with an integer k and asked whether there exists a collection of k mutually vertex-disjoint paths of length k in G. This problem is known to be fixed parameter tractable [11] and is easy to see that for this problem the "disjoint union" trick discussed in Section 5.1 does not directly apply. Thus we resort to polynomial parameter transformations.

Theorem 5.2.3 PATH PACKING *does not admit a polynomial kernel unless* PH=Σ_p^3.

Proof. We give a polynomial parameter transformation from the LONGEST PATH problem. Given an instance (G, k) to LONGEST PATH we construct a graph G' from G by adding $k-1$ vertex disjoint paths of length k. Now G contains a path of length k if and only if G' contains k paths of length k. This concludes the proof. ∎

Part II

New Methods

Chapter 6

Algorithmic Method: Chromatic Coding

In this chapter we present a subexponential-time algorithm for the k-Weighted Feedback Arc Set in Tournaments problem, defined in the next section. Our algorithm is based on a novel version of the Color Coding technique initiated in [11] and described in Section 2.7. The basic idea of Color Coding is to randomly color the input graph in order to make the solution more visible. That is, it is argued that with sufficiently high probability the random coloring unveils a structure that makes it easier to solve the problem instance. However, the more helpful we require the coloring to be, the less likely it is that a random coloring will be useful for our purposes. In the original Color Coding algorithm of Alon et al. [11] one required the coloring to color all vertices of the solution with different colors. Subsequently, the *Divide and Color* paradigm [95, 32] was introduced, and here the coloring was required only to split the solution in half in a specific way, thus making it possible to speed up some of the algorithms of Alon et al. [11].

In the algorithm we present, the set of usable colorings is much larger. We consider the solution set as a k-edge graph and require the random coloring to *properly color* this graph, that is the endpoints of every edge in this graph are colored with different colors. We call this variation of the Color Coding technique *Chromatic Coding.* Our algorithm runs in subexponential time, a trait uncommon to parameterized algorithms. In fact, to the author's best knowledge the only parameterized problems for which non-trivial subexponential time algorithms are known are bidimensional problems in planar graphs or graphs excluding a certain fixed graph H as a minor [43, 46, 49].

In order to derandomize our algorithm we construct a new kind of universal hash functions, that we coin *universal coloring families.* For integers m, k and r, a family $\mathcal{F}$ of functions from $[m]$ to $[r]$ is called a universal (m, k, r)-coloring family if for any graph G on the set of vertices $[m]$ with at most k edges, there exists an $f \in \mathcal{F}$ which is a proper vertex coloring of G. In the last section of the paper we give an explicit construction of a $(10k^2, k, O(\sqrt{k}))$-coloring family $\mathcal{F}$ of size $|\mathcal{F}| \leq 2^{\tilde{O}(\sqrt{k})}$ and an explicit universal $(n, k, O(\sqrt{k}))$-coloring family $\mathcal{F}$ of size $|\mathcal{F}| \leq 2^{\tilde{O}(\sqrt{k})} \log n$. We believe that these constructions can turn out to be useful to solve other edge subset problems in dense structures.

6.1 Tournament Feedback Arc Set

In a competition where everyone plays against everyone it is uncommon that the results are acyclic and hence one cannot rank the players by simply using a topological ordering. A natural ranking is one that minimizes the number of upsets, where an upset is a pair of players such that the lower ranked player beats the higher ranked one. The problem of finding such a ranking given the match outcomes is the FEEDBACK ARC SET problem restricted to tournaments. A *tournament* is a directed graph where every pair of vertices is connected by exactly one arc, and a *feedback arc set* is a set of arcs whose removal makes the graph acyclic. Feedback arc sets in tournaments are well studied, both from the combinatorial [56, 64, 86, 89, 90, 124, 129, 130, 137], statistical [127] and algorithmic [3, 6, 33, 91, 133, 134] points of view.

Unfortunately, the problem of finding a feedback arc set of minimum size in an unweighted tournament is NP-hard [6]. However, even the weighted version of the problem admits a polynomial time approximation scheme [91] and has been shown to be fixed parameter tractable [119]. One should note that the weighted generalization shown to admit a PTAS in [91] differs slightly from the one considered in this thesis. We consider the following problem:

> k-WEIGHTED FEEDBACK ARC SET IN TOURNAMENTS (k-FAST)
> INSTANCE: A tournament $T = (V, A)$, a weight function $w : A \to \{x \in \mathbb{R} : x \geq 1\}$ and an integer k.
> QUESTION: Is there an arc set $S \subseteq A$ such that $\sum_{e \in S} w(e) \leq k$ and $T \setminus S$ is acyclic?

The fastest previously known parameterized algorithm for k-FAST by Raman and Saurabh [119] runs in time $O(2.415^k \cdot k^{4.752} + n^{O(1)})$, and it was an open problem of Guo et al. [76] whether k-FAST can be solved in time $2^k \cdot n^{O(1)}$. We give a randomized and a deterministic algorithm both running in time $2^{O(\sqrt{k} \log^2 k)} + n^{O(1)}$.

6.1.1 Color and Conquer

Our algorithm consists of three steps. In the first step we reduce the instance to a problem kernel with at most $O(k^2)$ vertices, showing how to efficiently reduce the input tournament into one with $O(k^2)$ vertices, so that the original tournament has a feedback arc set of weight at most k, if and only if the new one has such a

1. Perform a data reduction to obtain a tournament T' of size $O(k^2)$.
2. Let $t = \sqrt{8k}$. Color the vertices of T' uniformly at random with colors from $\{1, \ldots, t\}$.
3. Let A_c be the set of arcs whose endpoints have different colors. Find a minimum weight feedback arc set contained in A_c, or conclude that no such feedback arc set exists.

Figure 6.1: Outline of the algorithm for k-FAST.

set. In the second step we randomly color the vertices of our graph with $t = \sqrt{8k}$ colors, and define the arc set A_c to be the set of arcs whose endpoints have different colors. In the last step the algorithm checks whether there is a weight k feedback arc set $S \subseteq A_c$. A summary of the algorithm is given in Figure 6.1.

6.1.2 Kernelization

For the first step of the algorithm we use the kernelization algorithm provided by Dom et al. [48]. They only show that the data reduction is feasible for the unweighted case, while in fact, it works for the weighted case as well. For completeness we provide a short proof of this. A triangle in what follows means a directed cyclic triangle.

Lemma 6.1.1 k-FAST *has a kernel with* $O(k^2)$ *vertices.*

Proof. We give two simple reduction rules.

1. If an arc e is contained in at least $k+1$ triangles reverse the arc and reduce k by $w(e)$.
2. If a vertex v is not contained in any triangle, delete v from T.

The first rule is safe because any feedback arc set that does not contain the arc e must contain at least one arc from each of the $k+1$ triangles containing e and thus must have weight at least $k+1$. The second rule is safe because the fact that v is not contained in any triangle implies that all arcs between $N^-(v)$ and $N^+(v)$ are oriented from $N^-(v)$ to $N^+(v)$. Hence for any feedback arc set S_1 of $T[N^-(v)]$ and feedback arc set S_2 of $T[N^+(v)]$, $S_1 \cup S_2$ is a feedback arc set of T.

Finally we show that any reduced yes instance T has at most $k(k+2)$ vertices. Let S be a feedback arc set of T with weight at most k. The set S contains at most k arcs, and for every arc $e \in S$, aside from the two endpoints of e, there are at most k vertices that are contained in a triangle containing e, because otherwise the first rule would have applied. Since every triangle in T contains an arc of S and every vertex of T is in a triangle, T has at most $k(k+2)$ vertices. ∎

6.1.3 Probability of a Good Coloring

We now proceed to analyze the second step of the algorithm. What we aim for, is to show that if T does have a feedback arc set S of weight at most k, then the probability that S is a subset of A_c is at least $2^{-c\sqrt{k}}$ for some fixed constant c. We show this by showing that if we randomly color the vertices of a k edge graph G with $t = \sqrt{8k}$ colors, then the probability that G has been properly colored is at least $2^{-c\sqrt{k}}$.

Lemma 6.1.2 *If a graph on q edges is colored randomly with $\sqrt{8q}$ colors then the probability that G is properly colored is at least $(2e)^{-\sqrt{q/8}}$.*

Proof. Arrange the vertices of the graph by repeatedly removing a vertex of lowest degree. Let $d_1, d_2, \ldots, d_s$ be the degrees of the vertices when they have been removed. Then for each i, $d_i(s-i+1) \leq 2q$, since when vertex i is removed each vertex had degree at least d_i. Furthermore, $d_i \leq s-i$ for all i, since the degree of the vertex removed can not exceed the number of remaining vertices at that point. Thus $d_i \leq \sqrt{2q}$ for all i. In the coloring, consider the colors of each vertex one by one starting from the last one, that is vertex number s. When vertex number i is colored, the probability that it will be colored by a color that differs from all those of its d_i neighbors following it is at least $(1 - \frac{d_i}{\sqrt{8q}}) \geq (2e)^{-d_i/\sqrt{8q}}$ because $\sqrt{8q} \geq 2d_i$. Hence the probability that G is properly colored is at least

$$\prod_{i=1}^{s}(1 - \frac{d_i}{\sqrt{8q}}) \geq \prod_{i=1}^{s}(2e)^{-d_i/\sqrt{8q}} = (2e)^{-\sqrt{q/8}}.$$

∎

6.1.4 Solving a Colored Instance

Given a t-colored tournament T, we will say that an arc set F is colorful if no arc in F is monochromatic. An ordering σ of T is colorful if the feedback arc set corresponding to σ is colorful. An optimal colorful ordering of T is a colorful ordering of T with minimum cost among all colorful orderings. We now give an algorithm that takes a t-colored arc weighted tournament T as input and finds a colorful feedback arc set of minimum weight, or concludes that no such feedback arc set exists.

Observation 6.1.3 *Let $T = (V_1 \cup V_2 \cup \ldots \cup V_t, A)$ be a t-colored tournament. There exists a colorful feedback arc set of T if and only if $T[V_i]$ induces an acyclic tournament for every i.*

We say that a colored tournament T is *feasible* if $T[V_i]$ induces an acyclic tournament for every i. Let $n_i = |V_i|$ for every i and let $\hat{n}$ be the vector $[n_1, n_2 \ldots n_t]$. Let $\sigma = v_1 v_2 \ldots v_n$ be the ordering of V corresponding to a colorful feedback arc set F of T. For every color class V_i of T, let $v_i^1 v_i^2 \ldots v_i^{n_i}$ be the order in which the vertices of V_i appear according to σ. Observe that since F is colorful, $v_i^1 v_i^2 \ldots v_i^{n_i}$ must be the unique topological ordering of $T[V_i]$. We exploit this to give a dynamic programming algorithm for the problem.

Lemma 6.1.4 *Given a feasible t-colored tournament T, we can find a minimum weight colorful feedback arc set in $O(t \cdot n^{t+1})$ time and $O(n^t)$ space.*

Proof. For an integer $x \geq 1$, define $S_x = \{v_1 \ldots, v_x\}$ and $S_x^i = \{v_1^i \ldots, v_x^i\}$. Let $S_0 = S_0^i = \emptyset$. Notice that for any x there must be some x' such that $S_x \cap V_i = S_{x'}$. Given an integer vector $\hat{p}$ of length t in which the ith entry is between 0 and n_i, let $T(\hat{p})$ be $T[S_{p_1}^1 \cup S_{p_2}^2 \ldots \cup S_{p_t}^t]$.

For a feasible t-colored tournament T, let $\textsc{Fas}(T)$ be the weight of the minimum weight colorful feedback arc set of T. Observe that if a t-colored tournament T is feasible then so are all induced subtournaments of T, and hence the function $\textsc{Fas}$ is well defined on all induced subtournaments of T. We proceed to prove that the following recurrence holds for $\textsc{Fas}(T(\hat{p}))$.

$$\textsc{Fas}(T(\hat{p})) = \min_{i \,:\, \hat{p}_i > 0} (\textsc{Fas}(T(\hat{p} - \hat{e}_i)) + \sum_{u \in V(T(\hat{p}))} w^*(v_{\hat{p}_i}^i, u)) \tag{6.1}$$

First we prove that the left hand side is at most the right hand side. Let i be the integer that minimizes the right hand side. Taking the optimal ordering of $T(\hat{p} - \hat{e}_i)$ and appending it with $v_{\hat{p}_i}^i$ gives an ordering of $T(\hat{p})$ with cost at most $\textsc{Fas}(T(\hat{p} - \hat{e}_i)) + \sum_{u \in V(T(\hat{p}))} w^*(v_{\hat{p}_i}^i, u)$.

To prove that the right hand side is at most the left hand side, take an optimal colorful ordering σ of $T(\hat{p})$ and let v be the last vertex of this ordering. There is an i such that $v = v_{\hat{p}_i}^i$. Thus σ restricted to $V(T(\hat{p} - \hat{e}_i))$ is a colorful ordering of $T(\hat{p} - \hat{e}_i)$ and the total weight of the edges with startpoint in v and endpoint in $V(T(\hat{p} - \hat{e}_i))$ is exactly $\sum_{u \in V(T(\hat{p}))} w^*(v_{\hat{p}_i}^i, u)$. Thus the cost of σ is at least the value of the right hand side of the inequality, completing the proof.

Recurrence 6.1 naturally leads to a dynamic programming algorithm for the problem. We build a table containing $\textsc{Fas}(T(\hat{p}))$ for every $\hat{p}$. There are $O(n^t)$ table entries, for each entry it takes $O(nt)$ time to compute it giving the $O(t \cdot n^{t+1})$ time bound. ■

In fact, the algorithm provided in Lemma 6.1.4 can be made to run slightly faster by pre-computing the value of $\sum_{u \in V(T(\hat{p}))} w^*(v^i_{\hat{p}_i}, u))$ for every $\hat{p}$ and i using dynamic programming, and storing it in a table. This would let us reduce the time to compute a table entry using Recurrence 6.1 from $O(nt)$ to $O(t)$ yielding an algorithm that runs in time and space $O(t \cdot n^t)$.

Lemma 6.1.5 k-FAST *(for a tournament of size $O(k^2)$) can be solved in expected time $2^{O(\sqrt{k}\log k)}$ and $2^{O(\sqrt{k}\log k)}$ space.*

Proof. Our algorithm proceeds as described in Figure 6.1. The correctness of the algorithm follows from Lemma 6.1.4. Combining Lemmata 6.1.1, 6.1.2, 6.1.4 yields an expected running time of $O((2e)^{\sqrt{k/8}}) \cdot O(\sqrt{8k} \cdot (k^2 + 2k)^{1+\sqrt{8k}}) \leq 2^{O(\sqrt{k}\log k)}$ for finding a feedback arc set of weight at most k if one exists. The space required by the algorithm is $O((k^2 + 2k)^{1+\sqrt{8k}}) \leq 2^{O(\sqrt{k}\log k)}$. ∎

The dynamic programming algorithm from Lemma 6.1.4 can be turned into a divide and conquer algorithm that runs in polynomial space, at a small cost in the running time.

Lemma 6.1.6 *Given a feasible t-colored tournament T, we can find a minimum weight colorful feedback arc set in time $O(n^{1+(t+2)\cdot\log n})$ in polynomial space.*

Proof. By expanding Recurrence (6.1) $\lfloor n/2 \rfloor$ times and simplifying the right hand side we obtain the following recurrence.

$$\textsc{Fas}(T(\hat{p})) = \min_{\substack{\hat{q} \geq \hat{0} \\ \hat{q}^\dagger \cdot \hat{e} = \lceil n/2 \rceil}} \{\textsc{Fas}(T(\hat{q})) + \textsc{Fas}(T \setminus V(T(\hat{q}))) + \sum_{\substack{u \in V(T(\hat{q})) \\ v \notin V(T(\hat{q}))}} w^*(v,u)\} \tag{6.2}$$

Recurrence 6.2 immediately yields a divide and conquer algorithm for the problem. Let $\mathcal{T}(n)$ be the running time of the algorithm restricted to a sub-tournament of T with n vertices. For a particular vector $\hat{q}$ it takes at most n^2 time to find the value of $\sum_{u \in V(T(\hat{q})), v \notin V(T(\hat{q}))} w^*(v,u)$. It follows that $\mathcal{T}(n) \leq n^{t+2} \cdot 2 \cdot \mathcal{T}(n/2) \leq 2^{\log n} \cdot n^{(t+2)\cdot\log n} = n^{1+(t+2)\cdot\log n}$. ∎

Theorem 6.1.7 k-FAST *(for a tournament of size $O(k^2)$) can be solved in expected time $2^{O(\sqrt{k}\log^2 k)}$ and polynomial space. Therefore,* k-FAST *for a tournament of size n can be solved in expected time $2^{O(\sqrt{k}\log^2 k)} + n^{O(1)}$ and polynomial space.*

6.2 Universal Coloring Families

For integers m, k and r, a family $\mathcal{F}$ of functions from $[m]$ to $[r]$ is called a universal (m, k, r)-coloring family if for any graph G on the set of vertices $[m]$ with at most k edges, there exists an $f \in \mathcal{F}$ which is a proper vertex coloring of G. An explicit construction of a $(10k^2, k, O(\sqrt{k}))$-coloring family can replace the randomized coloring step in the algorithm for k-FAST. In this section, we provide such a construction.

Theorem 6.2.1 *There exists an explicit universal* $(10k^2, k, O(\sqrt{k}))$*-coloring family* $\mathcal{F}$ *of size* $|\mathcal{F}| \leq 2^{\tilde{O}(\sqrt{k})}$.

Proof. For simplicity we omit all floor and ceiling signs whenever these are not crucial. We make no attempt to optimize the absolute constants in the $\tilde{O}(\sqrt{k})$ or in the $O(\sqrt{k})$ notation. Whenever this is needed, we assume that k is sufficiently large.

Let $\mathcal{G}$ be an explicit family of functions g from $[10k^2]$ to $[\sqrt{k}]$ so that every coordinate of g is uniformly distributed in $[\sqrt{k}]$, and every two coordinates are pairwise independent. There are known constructions of such a family $\mathcal{G}$ with $|\mathcal{G}| \leq k^{O(1)}$. Indeed, each function g represents the values of $10k^2$ pairwise independent random variables distributed uniformly in $[\sqrt{k}]$ in a point of a small sample space supporting such variables; a construction is given, for example, in [7]. The family $\mathcal{G}$ is obtained from the family of all linear polynomials over a finite field with some $k^{O(1)}$ elements, as described in [7].

We can now describe the required family $\mathcal{F}$. Each $f \in \mathcal{F}$ is described by a subset $T \subset [10k^2]$ of size $|T| = \sqrt{k}$ and by a function $g \in \mathcal{G}$. For each $i \in [10k^2]$, the value of $f(i)$ is determined as follows. Suppose $T = \{i_1, i_2, \ldots, i_{\sqrt{k}}\}$, with $i_1 < i_2 < \ldots < i_{\sqrt{k}}$. If $i = i_j \in T$, define $f(i) = \sqrt{k} + j$. Otherwise, $f(i) = g(i)$. Note that the range of f is of size $\sqrt{k} + \sqrt{k} = 2\sqrt{k}$, and the size of $\mathcal{F}$ is at most

$$\binom{10k^2}{\sqrt{k}}|\mathcal{G}| \leq \binom{10k^2}{\sqrt{k}} k^{O(1)} \leq 2^{O(\sqrt{k}\log k)} \leq 2^{\tilde{O}(\sqrt{k})}.$$

To complete the proof we have to show that for every graph G on the set of vertices $[10k^2]$ with at most k edges, there is an $f \in \mathcal{F}$ which is a proper vertex coloring of G. Fix such a graph G.

The idea is to choose T and g in the definition of the function f that will provide the required coloring for G as follows. The function g is chosen at random in $\mathcal{G}$, and is used to properly color all but at most $\sqrt{k}$ edges. The set T is chosen to contain at least one endpoint of each of these edges, and the vertices in the set T will be re-colored by a unique color that is used only once by f. Using the properties of $\mathcal{G}$ we now observe that with positive probability the number of edges of G which are monochromatic is bounded by $\sqrt{k}$.

Claim 6.2.2 *If the vertices of G are colored by a function g chosen at random from $\mathcal{G}$, then the expected number of monochromatic edges is $\sqrt{k}$.*

Proof. Fix an edge e in the graph G and $j \in [\sqrt{k}]$. As g maps the vertices in a pairwise independent manner, the probability that both the end points of e get mapped to j is precisely $\frac{1}{(\sqrt{k})^2}$. There are $\sqrt{k}$ possibilities for j and hence the probability that e is monochromatic is given by $\frac{\sqrt{k}}{(\sqrt{k})^2} = \frac{1}{\sqrt{k}}$. Let X be the random variable denoting the number of monochromatic edges. By linearity of expectation, the expected value of X is $k \cdot \frac{1}{\sqrt{k}} = \sqrt{k}$. ∎

Returning to the proof of the theorem, observe that by the above claim, with positive probability, the number of monochromatic edges is upper bounded by $\sqrt{k}$. Fix a $g \in \mathcal{G}$ for which this holds and let $T = \{i_1, i_2, \ldots, i_{\sqrt{k}}\}$ be a set of $\sqrt{k}$ vertices containing at least one endpoint of each monochromatic edge. Consider the function f defined by this T and g. As mentioned above f colors each of the vertices in T by a unique color, which is used only once by f, and hence we only need to consider the coloring of $G \setminus T$. However all edges in $G \setminus T$ are properly colored by g and f coincides with g on $G \setminus T$. Hence f is a proper coloring of G, completing the proof of the theorem. ∎

Remarks:

- Each universal $(n, k, O(\sqrt{k}))$-coloring family must also be an $(n, \sqrt{k}, O(\sqrt{k}))$-hashing family, as it must contain, for every set S of $\sqrt{k}$ vertices in $[n]$, a function that maps the elements of S in a one-to-one manner, since these vertices may form a clique that has to be properly colored by a function of the family. Therefore, by the known bounds for families of hash functions (see, e.g., [117]), each such family must be of size at least $2^{\tilde{\Omega}(\sqrt{k})} \log n$.

Although the next result is not required for our results on the Tournament Feedback Arc Set problem, we present it here as it may be useful in similar applications.

Theorem 6.2.3 *For any $n > 10k^2$ there exists an explicit universal $(n, k, O(\sqrt{k}))$-coloring family $\mathcal{F}$ of size $|\mathcal{F}| \leq 2^{\tilde{O}(\sqrt{k})} \log n$.*

Proof. Let $\mathcal{F}_1$ be an explicit $(n, 2k, 10k^2)$-family of hash functions from $[n]$ to $10k^2$ of size $|\mathcal{F}_1| \leq k^{O(1)} \log n$. This means that for every set $S \subset [n]$ of size at most $2k$ there is an $f \in \mathcal{F}_1$ mapping S in a one-to-one fashion. The existence of such a family is well known, and follows, for example, from constructions of small spaces supporting n nearly pairwise independent random variables taking values in $[10k^2]$. Let $\mathcal{F}_2$ be an explicit universal $(10k^2, k, O(\sqrt{k}))$-coloring family, as described in Theorem 6.2.1. The required family $\mathcal{F}$ is simply the family of all compositions of a function from $\mathcal{F}_2$ followed by one from $\mathcal{F}_1$. It is easy to check that $\mathcal{F}$ satisfies the assertion of Theorem 6.2.3. ■

Finally, combining the algorithm from Theorem 6.1.7 with the universal coloring family given by Theorem 6.2.1 yields a deterministic subexponential time polynomial space algorithm for k-FAST.

Theorem 6.2.4 k-FAST *can be solved in time* $2^{\tilde{O}(\sqrt{k})} + n^{O(1)}$ *and polynomial space.*

Chapter 7

Explicit Identification in W[1]-Hardness Reductions

In hardness reductions one has to code one combinatorial problem in the language of another problem. To this end we require *gadgets*, small constructions that act as "subroutines" in our encoding. Typically one needs to construct gadgets that encode selection of an element from a specific set, and gadgets that pass around information on which elements that have been selected. How this information is encoded greatly affects how the corresponding gadgets have to behave. In Sections 7.1.1 and 7.1.2 we show that sometimes it is useful to assign identification numbers to the elements of the sets we are working with, because numbers often can be encoded in a combinatorial problem as a distribution of resources. In Section 7.2 we show that the results from Sections 7.1.1 and 7.1.2 can be used to prove an interesting trade-off between expressive power and computational tractability.

7.1 Hardness for Treewidth-Parameterizations

Due to Courcelle's Theorem (Theorem 2.4.1) many problems admit FPT algorithms on graphs of bounded treewidth. It is natural to ask whether all problems that can be solved in polynomial time on graphs with constant treewidth admit FPT algorithms when parameterized by the treewidth of the input graph. That is, are all problems that are solvable in time $n^{f(\mathbf{tw}(G))}$ also solvable in time $g(\mathbf{tw}(G))n^{O(1)}$? We show that the answer to this question is no, unless FPT = W[1]. In particular, we show that the EQUITABLE COLORING and CAPACITATED DOMINATING SET problems are W[1]-hard parameterized by the treewidth of the input graph. These were the first problems to be shown W[1]-hard parameterized by the treewidth of the input graph.

7.1.1 Equitable Coloring

The notion of equitable coloring seems to have been first introduced by Meyer in 1973, where an application to scheduling garbage trucks is described [109]. Recently, Bodlaender and Fomin have shown that determining whether a graph of treewidth at most t admits an equitable coloring, can be solved in time $O(n^{O(t)})$ [21].

We consider the parameterized complexity of EQUITABLE COLORING (ECP) in graphs with bounded treewidth. We show that when ECP is parameterized by (t, r), where t is the treewidth bound, and r is the number of color classes, the problem is W[1]-hard. We reduce from the MULTICOLOR CLIQUE problem defined in Section 3.2. Our reduction is based on a methodology which is sometimes termed as *edge representation strategy.* This strategy is very basic and is useful for many reductions. Consider that the instance $G = (V, E)$ of MULTICOLOR CLIQUE has its vertices colored by the integers $1, ..., k$. Let $V[i]$ denote the set of vertices of color i, and let $E[i, j]$, for $1 \leq i < j \leq k$, denote the set of edges $e = uv$, where $u \in V[i]$ and $v \in V[j]$. We also assume that $|V[i]| = N$ for all i, and that $|E[i, j]| = M$ for all $i < j$, that is, the vertex color classes of G, and also the edge sets between them, have uniform sizes. See Section 3.2 for a justification of these assumptions. In this methodology our basic encoding gadgets correspond to edges which we call *edge gadget.* We generally have three kind of gadgets which are engineered together to get a reduction for the problem.

Selection Gadget: This gadget job is to select exactly one edge gadget among edge gadgets corresponding to edges between any two color classes $V[i]$ and $V[j]$.

Coherence Gadget: This gadget makes sure that the edge gadgets selected among edge gadgets corresponding to edges emanating out from a particular color class $V[i]$ has a vertex in common in $V[i]$. That is all the edges corresponding to selected edge gadgets emanates from the same vertex in $V[i]$.

Match Gadget: This gadget ensures that if we have selected an edge gadget corresponding to an edge (u, v) between $V[i]$ and $V[j]$ then the edge gadget selected between $V[j]$ and $V[i]$ corresponds to (v, u).

In what follows next we show how to adhere to this strategy and form gadgets in the context of reduction from MULTICOLOR CLIQUE to EQUITABLE COLORING PROBLEM in graphs with bounded treewidth. To show the desired reduction, we introduce two more general problems. List analogues of equitable coloring have been previously studied by Kostochka, et al. [1].

LIST EQUITABLE COLORING PROBLEM (LECP): Given an input graph $G = (V, E)$, lists L_v of colors for every vertex $v \in V$ and a positive integer r; does there exist a proper coloring f of G with r colors that for every vertex $v \in V$ uses a color from its list L_v such that for any two color class, V_i and V_j of the coloring f, $||V_i| - |V_j|| \leq 1$?

Number List Coloring Problem (NLCP): Given an input graph $G = (V, E)$, lists L_v of colors for every vertex $v \in V$, a function $h : \cup_{v \in V} L_v \to \mathbb{N}$, associating a number to each color, and a positive integer r; does there exist a proper coloring f of G with r colors that for every vertex $v \in V$ uses a color from its list L_v, such that any color class V_c of the coloring f is of size $h(c)$?

Our main effort is in the reduction of the Multicolor Clique problem to NLCP. We will use the following sets of colors in our construction of an instance of NLCP:

1. $\mathcal{S} = \{\sigma[i,j] : 1 \leq i \neq j \leq k\}$
2. $\mathcal{S}' = \{\sigma'[i,j] : 1 \leq i \neq j \leq k\}$
3. $\mathcal{T} = \{\tau_i[r,s] : 1 \leq i \leq k,\ 1 \leq r < s \leq k, r \neq i, s \neq i\}$
4. $\mathcal{T}' = \{\tau_i'[r,s] : 1 \leq i \leq k,\ 1 \leq r < s \leq k, r \neq i, s \neq i\}$
5. $\mathcal{E} = \{\epsilon[i,j] : 1 \leq i < j \leq k\}$
6. $\mathcal{E}' = \{\epsilon'[i,j] : 1 \leq i < j \leq k\}$

Note that $|\mathcal{S}| = |\mathcal{S}'| = 2\binom{k}{2}$, that is, there are distinct colors $\sigma[2,3]$ and $\sigma[3,2]$, etc. In contrast, the colors $\tau_i[r,s]$ are only defined for $r < s$. We associate with each vertex and edge of G a pair of (unique) *identification numbers*. The *up-identification number* $v[up]$ for a vertex v should be in the range $[n^2+1, n^2+n]$, if G has n vertices and it could be chosen arbitrarily, but uniquely. Similarly, the *up-identification number* $e[up]$ of an edge e of G can be assigned (arbitrarily, but uniquely) in the range $[2n^2+1, 2n2+m]$, assuming G has m edges.

Choose a suitably large positive integer Z_0, for example $Z_0 = n^3$, and define the *down-identification number* $v[down]$ for a vertex v to be $Z_0 - v[up]$, and similarly for the edges e of G, define the *down-identification number* $e[down]$ to be $Z_0 - e[up]$. Choose a second large positive integer, $Z_1 >> Z_0$, for example, we may take $Z_1 = n^6$.

Next we describe various gadgets and the way they are combined in the reduction. First we describe the gadget which encodes the *selection* of the edge going between two particular color classes in G. In other words, we will think of the representation of a k-clique in G as involving the selection of edges (with each edge selected twice, once in each direction) between the color classes of vertices in G, with gadgets for *selection*, and to check two things: (1) that the selections in opposite color directions match, and (2) that the edges chosen from color class $V[i]$ going to $V[j]$ (for various $j \neq i$) all emanate from the same vertex in $V[i]$.

There are $2\binom{k}{2}$ groups of gadgets, one for each pair of color indices $i \neq j$. If $1 \leq i < j \leq k$, then we will refer to the gadgets in the group $\mathcal{G}[i,j]$ as *forward gadgets*, and we will refer to the gadgets in the group $\mathcal{G}[j,i]$ as *backward gadgets.*

If $e \in E[i,j]$, then there is one forward gadget corresponding to e in the group $\mathcal{G}[i,j]$, and one backward gadget corresponding to e in the group $\mathcal{G}[j,i]$. The construction of these gadgets is described as follows.

The forward gadget corresponding to $e = uv \in E[i,j]$.
The gadget has a root vertex $r[i,j,e]$, and consists of a tree of height 2. The list assigned to this root vertex contains two colors: $\sigma[i,j]$ and $\sigma'[i,j]$. The root vertex has $Z_1 + 1$ children, and each of these is also assigned the two-element list containing the colors $\sigma[i,j]$ and $\sigma'[i,j]$. One of the children vertices is distinguished, and has $2(k-1)$ groups of further children:

- $e[up]$ children assigned the list $\{\sigma'[i,j], \epsilon[i,j]\}$.
- $e[down]$ children assigned the list $\{\sigma'[i,j], \epsilon'[i,j]\}$.
- For each r in the range $j < r \leq k$, $u[up]$ children assigned the list $\{\sigma'[i,j], \tau_i[j,r]\}$.
- For each r in the range $j < r \leq k$, $u[down]$ children assigned $\{\sigma'[i,j], \tau'_i[j,r]\}$.
- For each r in the range $1 \leq r < j$, $u[down]$ children assigned $\{\sigma'[i,j], \tau_i[r,j]\}$.
- For each r in the range $1 \leq r < j$, $u[up]$ children assigned the list $\{\sigma'[i,j], \tau'_i[r,j]\}$.

The backward gadget corresponding to $e = uv \in E[i,j]$.
The gadget has a root vertex $r[j,i,e]$, and consists of a tree of height 2. The list assigned to this root vertex contains two colors: $\sigma[j,i]$ and $\sigma'[j,i]$. The root vertex has $Z_1 + 1$ children, and each of these is also assigned the two-element list containing the colors $\sigma[j,i]$ and $\sigma'[j,i]$. One of the children vertices is distinguished, and has $2k$ groups of further children:

- $e[up]$ children assigned the list $\{\sigma'[j,i], \epsilon'[i,j]\}$.
- $e[down]$ children assigned the list $\{\sigma'[j,i], \epsilon[i,j]\}$.
- For each r in the range $i < r \leq k$, $v[up]$ children assigned the list $\{\sigma'[j,i], \tau_j[i,r]\}$.
- For each r in the range $i < r \leq k$, $v[down]$ children assigned $\{\sigma'[j,i], \tau'_j[i,r]\}$.
- For each r in the range $1 \leq r < i$, $v[down]$ children assigned $\{\sigma'[j,i], \tau_j[r,i]\}$.
- For each r in the range $1 \leq r < i$, $v[up]$ children assigned the list $\{\sigma'[j,i], \tau'_j[r,i]\}$.

The numerical targets (h function) .

1. For all $c \in (\mathcal{T} \cup \mathcal{T}')$, $h(c) = Z_0$.
2. For all $c \in (\mathcal{E} \cup \mathcal{E}')$, $h(c) = Z_0$.
3. For all $c \in \mathcal{S}$, $h(c) = (M-1)(Z_1+1)+1$.
4. For all $c \in \mathcal{S}'$, $h(c) = (M-1)+(Z_1+1)+(k-1)(M-1)Z_0$.

That completes the formal description of the reduction from Multicolor Clique to NLCP. We turn now to some motivating remarks about the design of the reduction.

Remarks on the colors, their numerical targets, and their role in the reduction.

(**1**). There are $2\binom{k}{2}$ groups of gadgets. Each edge of G gives rise to two gadgets. Between any two color classes of G there are precisely M edges, and therefore $M \cdot \binom{k}{2}$ edges in G in total. Each group of gadgets therefore contains M gadgets. The gadgets in each group have two "helper" colors. For example, the group of gadgets $\mathcal{G}[4,2]$ has the helper colors $\sigma[4,2]$ and $\sigma'[4,2]$. The role of the gadgets in this group is to indicate a choice of an edge going *from* a vertex in the color class $V[4]$ of G *to* a vertex in the color class $V[2]$ of G. The role of the $2\binom{k}{2}$ groups of gadgets is to represent the selection of $\binom{k}{2}$ edges of G that form a k-clique, with each edge chosen twice, once in each direction. If $i < j$ then the choice is represented by the coloring of the gadgets in the group $\mathcal{G}[i,j]$, and these are the *forward* gadgets of the edge choice. If $j < i$, then the gadgets in $\mathcal{G}[i,j]$ are *backward* gadgets (representing the edge selection in the opposite direction, relative to the ordering of the color classes of G). The numerical targets for the colors in $\mathcal{S} \cup \mathcal{S}'$ are chosen to force exactly one edge to be selected (forward or backward) by each group of gadgets, and to force the gadgets that are colored in a way that indicates the edge was not selected into being colored in a particular way (else the numerical targets cannot be attained). The numerical targets for these colors are complicated, because of this role (which is asymmetric between the pair of colors $\sigma[i,j]$ and $\sigma'[i,j]$).

(**2**). The colors in $\mathcal{T} \cup \mathcal{T}'$ and $\mathcal{E} \cup \mathcal{E}'$ are organized in symmetric pairs, and each pair is used to transmit (and check) information. Due to the enforcements alluded to above, each "selection" coloring of a gadget (there will be only one possible in each group of gadgets) will force some number of vertices to be colored with these pairs of colors, which can be thought of as an information transmission. For example, when a gadget in $\mathcal{G}[4,2]$ is colored with a "selection" coloring, this indicates that the edge from which the gadget arises is selected as the edge *from* the color class $V[4]$ of G, *to* the color class $V[2]$. There is a pair of colors that handles the information transmission concerning *which edge is selected* between the groups

$\mathcal{G}[2,4]$ and $\mathcal{G}[4,2]$. (Of course, something has to check that the edge selected in one direction, is the same as the edge selected in the other direction.) There is something elegant about the dual-color transmission channel for this information. Each vertex and edge has two unique identification numbers, "up" and "down", that sum to Z_0. To continue the concrete example, $\mathcal{G}[4,2]$ uses the (number of vertices colored by the) pair of colors $\epsilon[2,4]$ and $\epsilon'[2,4]$ to communicate to $\mathcal{G}[2,4]$ about the edge selected. The signal from one side consists of $e[up]$ vertices colored $\epsilon[2,4]$ and $e[down]$ vertices colored $\epsilon'[2,4]$. The signal from the other side consists of $e[down]$ vertices colored $\epsilon[2,4]$ and $e[up]$ vertices colored $\epsilon'[2,4]$. Thus the numerical targets for these colors allow us to check whether the same edge has been selected in each direction (if each color target of Z_0 is met). There is the additional advantage that the *amount* of signal in each direction is the same: in each direction a total of Z_0 colored vertices, with the two paired colors, constitutes the signal. This means that, modulo the discussion in (1) above, when an edge is *not* selected, the corresponding non-selection coloring involves uniformly the same number (i.e., Z_0) of vertices colored "otherwise" for each of the $(M-1)$ gadgets colored in the non-selection way: this explains (part of) the $(k-1)(M-1)Z_0$ term in (4) of the numerical targets.

(**3**). In a similar manner to the communication task discussed above, each of the $k-1$ groups of gadgets $\mathcal{G}[i,\_]$ need to check that each has selected an edge *from* $V[i]$ that originates at the same vertex in $V[i]$. Hence there are pairs of colors that provide a communication channel similar to that in (2) for this information. This role is played by the colors in $\mathcal{T} \cup \mathcal{T}'$. (Because of the bookkeeping issues, this becomes somewhat intricate in the formal definition of the reduction.)

The above remarks are intended to aid an intuitive understanding of the reduction. We now return to a more formal argument.

Claim 7.1.1 *If G has a k-multicolor clique, then G' is a yes-instance to NLCP.*

Proof. The proof of this claim is relatively straightforward. The gadgets corresponding to the edges of a k-clique in G are colored in a manner that indicates "selected" (for both the forward and the backward gadgets) and all other gadgets are colored in manner that indicates "non-selected". The coloring that corresponds to "selected" colors the root vertex with the color $\sigma[i,j]$, and this forces the rest of the coloring of the gadget. The coloring that corresponds to "non-selected" colors the root vertex with the color $\sigma'[i,j]$. In this case the coloring of the rest of the gadget is not entirely forced, but if the grandchildren vertices of the gadget are also colored with $\sigma'[i,j]$, then all the numerical targets will be met. ∎

Claim 7.1.2 *Suppose that Γ is a list coloring of G' that meets all the numerical targets. Then in each group of gadgets, exactly one gadget is colored in a way that indicates "selection".*

Proof. We argue this as follows. There cannot be two gadgets in any group colored in the "selection" manner, since this would make it impossible to meet the numerical target for a color in $\mathcal{S}$. If no gadget is colored in the "selection" manner, then again the targets cannot be met for the colors in $\mathcal{S} \cup \mathcal{S}'$ used in the lists for this group of gadgets. ∎

Claim 7.1.3 *Suppose that* Γ *is a list coloring of* G' *that meets all the numerical targets. Then in each group of gadgets, every gadget that is not colored in a way that indicates "selection" must have all of its grandchildren vertices colored with the appropriate color in* $\mathcal{S}'$*.*

Proof. This claim follows from Claim 7.1.2, noting that the numerical targets for the $\mathcal{S}'$ colors cannot be met unless this is so. ∎

It follows from Claims 7.1.2 and 7.1.3, that if Γ is a list coloring of G' that meets all the numerical targets, then in each group of gadgets, exactly one gadget is colored in the "selection" manner, and all other gadgets are colored in a completely determined "nonselection" manner. Each "selection" coloring of a gadget produces a numerical signal (based on vertex and edge identification numbers) carried by the colors in $\mathcal{T} \cup \mathcal{T}'$ and $\mathcal{E} \cup \mathcal{E}'$, with two signals per color. The target of Z_0 for these colors can only be achieved if the selection colorings indicate a clique in G.

Theorem 7.1.4 *NLCP is* W[1]*-hard for trees, parameterized by the number of colors that appear on the lists.*

The reduction from NLCP to LECP is almost trivial, achieved by padding with isolated vertices having single-color lists. The reduction from LECP to ECP is described as follows. Create a clique of size r, the number of colors occurring on the lists, and connect the vertices of this clique to the vertices of G' in a manner that enforces the lists. Since G' is a tree, the treewidth of the resulting graph is at most r. We have:

Theorem 7.1.5 Equitable Coloring *is* W[1]*-hard parameterized by treewidth and the number* r *of colors.*

7.1.2 Capacitated Dominating Set

In this section we show that Capacitated Dominating Set is $W[1]$-hard when parameterized by treewidth and solution size. Just as in the previous section, we reduce from Multicolor Clique. See Section 3.2 for the definition of this problem. In fact, we will reduce to a slightly modified version of Capacitated Dominating Set, Marked Capacitated Dominating Set where we *mark* some vertices and demand that all marked vertices must be in the dominating set. We can then reduce from Marked Capacitated Dominating Set to Capacitated Dominating Set by attaching $k+1$ leaves to each marked vertex and increasing the capacity of each marked vertex by $k+1$. It is easy to see that the new instance has a k-capacitated dominating set if and only if the original one had a k-capacitated dominating set that contained all marked vertices, and that this operation does not increase the treewidth of the graph. Thus, to prove that Capacitated Dominating Set is $W[1]$-hard when parameterized by treewidth and solution size, it is sufficient to prove that Marked Capacitated Dominating Set is. We will show how given an instance (G, k) of Multicolor Clique, we can build an instance (H, c, k') of Marked Capacitated Dominating Set such that

- $k' = 7k(k-1) + 2k$,
- G has a clique of size k if and only if H has a capacitated dominating set of size k', and
- the treewidth of H is $O(k^4)$.

We employ a strategy similar to the *edge representation* strategy we used to show that Equitable Coloring is W[1]-hard parameterized by treewidth. The main difference in our approach is that the coherence gadget is switched out with a *vertex selection* gadget and the vertex and edge selection gadgets are joined together in such a way that one only can pick an edge if one also picks its endpoints.

For a pair of distinct integers i, j, let $E[i, j]$ be the set of edges with one endpoint in $V[i]$ and the other in $V[j]$. Without loss of generality, we will assume that $|V[i]| = N$ and $|E[i, j]| = M$ for all i, j, $i \neq j$. To each vertex v we assign a unique identification number v^{up} between $N+1$ and $2N$, and we set $v^{down} = 2N - v^{up}$. For two vertices u and v, by adding an (A, B)-*arrow* from u to v we will mean adding A subdivided edges between u and v and attaching B leaves to v (see Fig. 7.1). Now we describe how to build the graph H for a given instance $(G = (V[1] \cup V[2] \cdots \cup V[k], E), k)$ of Multicolor Clique.

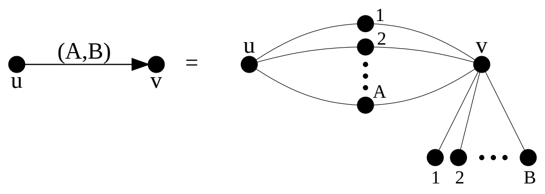

Figure 7.1: Adding an (A, B)-arrow from u to v.

For every integer i between 1 and k we add a marked vertex $\hat{x}_i$ that has a neighbor $\overline{v}$ for every vertex v in $V[i]$. For every $j \neq i$, we add a marked vertex $\hat{y}_{ij}$

and a marked vertex $\hat{z}_{ij}$. Now, for every vertex $v \in V[i]$ and every integer $j \neq i$ we add a (v^{up}, v^{down})-arrow from $\overline{v}$ to $\hat{y}_{ij}$ and a (v^{down}, v^{up})-arrow from $\overline{v}$ to $\hat{z}_{ij}$. Finally we add a set S_i of $k'+1$ vertices and make every vertex in S_i adjacent to every vertex $\overline{v}$ with $v \in V[i]$. See Fig. 7.2 for an illustration.

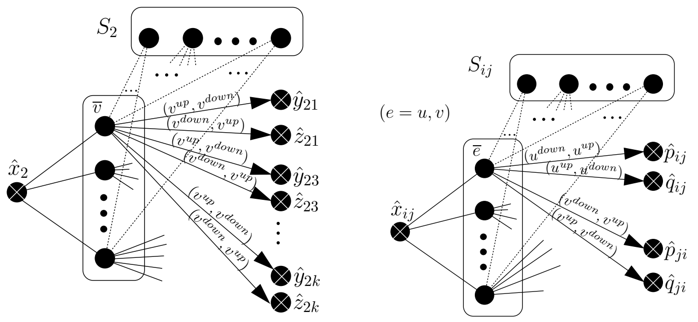

Figure 7.2: Vertex and edge selection gadgets.

Similarly, for every pair of integers i, j with $i < j$, we add a marked vertex $\hat{x}_{ij}$ with a neighbor $\overline{e}$ for every edge e in $E[i, j]$. Moreover, we add four new marked vertices $\hat{p}_{ij}, \hat{p}_{ji}, \hat{q}_{ij}$, and $\hat{q}_{ji}$. For every edge $e = \{u, v\}$ in $E[i, j]$ with $u \in V[i]$ and $v \in V[j]$, we add a (u^{down}, u^{up})-arrow from $\overline{e}$ to $\hat{p}_{ij}$, a (u^{up}, u^{down})-arrow from $\overline{e}$ to $\hat{q}_{ij}$, a (v^{down}, v^{up})-arrow from $\overline{e}$ to $\hat{p}_{ji}$ and a (v^{up}, v^{down})-arrow from $\overline{e}$ to $\hat{p}_{ji}$. We also add a set S_{ij} of $k'+1$ vertices and make every vertex in S_{ij} adjacent to every vertex $\overline{e}$ with $e \in E[i, j]$. See Fig. 7.2 for an illustration.

Finally, we add a marked vertex $\hat{r}_{ij}$ and a marked vertex $\hat{s}_{ij}$ for every $i \neq j$. For every $i \neq j$, we add $(2N, 0)$-arrows from $\hat{y}_{ij}$ to $\hat{r_{ij}}$, from $\hat{p}_{ij}$ to $\hat{r_{ij}}$, from $\hat{z}_{ij}$ to $\hat{s_{ij}}$, and from $\hat{q}_{ij}$ to $\hat{s_{ij}}$ (see Fig. 7.3). This concludes the description of the graph H.

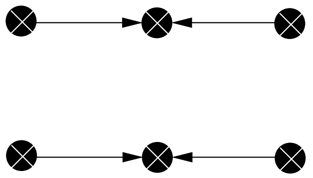

Figure 7.3: Vertex-Edge incidence gadget

We now describe the capacities of the vertices. For every $i \neq j$, the vertex $\hat{x}_i$ has capacity $N-1$, the vertex $\hat{x}_{ij}$ has capacity $M-1$, the vertices $\hat{y}_{ij}$ and $\hat{z}_{ij}$ both have capacity $2N^2$, the vertices $\hat{p}_{ij}$ and $\hat{q}_{ij}$ have capacity $2NM$, and both $\hat{r}_{ij}$ and $\hat{s}_{ij}$ have capacity $2N$. For all other vertices, their capacity is equal to their degree in H.

Observation 7.1.6 *The treewidth of H is $O(k^4)$.*

Proof. If we remove all marked vertices ($\bigcup_{i=1}^{k} S_i$ and $\bigcup_{i \neq j} S_{ij}$), a total of $O(k^4)$ vertices, from H, we obtain a forest. As deleting a vertex reduces the treewidth by at most one, this concludes the proof. ∎

Lemma 7.1.7 *If G has a multicolor clique $C = \{v_1, v_2, \ldots, v_k\}$ then H has a capacitated dominating set D of size k' containing all marked vertices.*

Proof. For every $i < j$ let e_{ij} be the edge from v_i to v_j in G. In addition to all the marked vertices, let D contain $\overline{v_i}$ and $\overline{e_{ij}}$ for every $i < j$. Clearly D contains exactly k' vertices, so it remains to prove that D is indeed a capacitated dominating set.For every $i < j$, let $\hat{x}_i$ and $\hat{x}_{ij}$ dominate all their neighbors except for $\overline{v_i}$ and $\overline{e_{ij}}$ respectively. The vertices $\overline{v_i}$ and $\overline{e_{ij}}$ can dominate all their neighbors, since their capacity is equal to their degree. Let $\hat{r}_{ij}$ dominate v_i^{down} of the vertices in the $(2N, 0)$-arrow from $\hat{y}_{ij}$, and v_i^{up} of the vertices of the $(2N, 0)$-arrow from $\hat{p}_{ij}$. Similarly let $\hat{s}_{ij}$ dominate v_i^{up} of the vertices in the $(2N, 0)$-arrow from $\hat{z}_{ij}$, and v_i^{down} of the vertices of the $(2N, 0)$-arrow from $\hat{q}_{ij}$. Finally, for every $i \neq j$ we let $\hat{y}_{ij}$, $\hat{z}_{ij}$, $\hat{p}_{ij}$ and $\hat{q}_{ij}$ dominate all their neighbors that have not been dominated yet. One can easily check that every vertex of H will either be a dominator or dominated in this manner, and that no dominator dominates more vertices than its capacity. ∎

Lemma 7.1.8 *If H has a capacitated dominating set D of size k' containing all marked vertices, then G has a multicolor clique of size k.*

Proof. Observe that for every integer $1 \leq i \leq k$, there must be a $v_i \in V[i]$ such that $\overline{v}_i \in D$. Otherwise we have that $S_i \subset D$ and, since $|S_i| > k'$, we obtain a contradiction. Similarly, for every pair of integers i, j with $i < j$ there must be an edge $e_{ij} \in E[i, j]$ such that $\overline{e}_{ij} \in D$. We let $e_{ji} = e_{ij}$. Since $|D| \leq k'$ it follows that these are the only unmarked vertices in D. Since all the unmarked vertices in D have capacity equal to their degree, we can assume that each such vertex dominates all its neighbors. We now proceed with proving that for every pair of integers i,j with $i \neq j$, $e_{ij} = uv$ is incident to v_i. We prove this by showing that if $u \in V[i]$ then $v_i^{up} + u^{down} = 2N$.

Suppose for a contradiction that $v_i^{up} + u^{down} < 2N$. Observe that each vertex of $T = (N(\hat{y}_{ij}) \cup N(\hat{r}_{ij}) \cup N(\hat{p}_{ij})) \setminus (N(\overline{v}_i) \cup N(\overline{e}_{ij}))$ must be dominated by either $\hat{y}_{ij}$, $\hat{r}_{ij}$, or $\hat{p}_{ij}$. However, by our assumption that $v_i^{up} + u^{down} < 2N$, it follows that $|T| = 2N^2 + 4N + 2MN - (v_i^{up} + u^{down}) > 2N^2 + 2N + 2MN$. The sum of the capacities of $\hat{y}_{ij}$, $\hat{r}_{ij}$, and $\hat{p}_{ij}$ is exactly $2N^2 + 2N + 2MN$. Thus it is impossible that every vertex of T is dominated by one of $\hat{y}_{ij}$, $\hat{r}_{ij}$, and $\hat{p}_{ij}$, a contradiction. If $v_i^{up} + u^{down} > 2N$ then $v_i^{down} + u^{up} < 2N$, and we can apply an identical argument for $\hat{z}_{ij}$, $\hat{s}_{ij}$, and $\hat{q}_{ij}$.

Thus, it follows that for every $i \neq j$ there is an edge e_{ij} incident both to v_i and to v_j. Thus $\{v_1, v_2, \ldots, v_k\}$ forms a clique in G. As any k-clique in G is a multicolor clique this completes the proof. ∎

Observation 7.1.6, Lemma 7.1.7 and Lemma 7.1.8 immediately imply the following theorem.

Theorem 7.1.9 Capacitated Dominating Set *parameterized by treewidth and solution size is* $W[1]$*-hard.*

7.2 Hardness for Cliquewidth-Parameterizations

By the seminal result of Courcelle [35] (Theorem 2.4.1), all problems expressible in MSO_2 logic are FPT when parameterized by the treewidth of the input graph. The result of Courcelle was extended by Courcelle, Makowsky and Rotics [36] (Theorem 2.4.4) who proved that all problems expressible in MSO_1 logic are FPT in parameterized by the cliquewidth of the input graph. For several of the problems expressible in MSO_2 but not MSO_1, like Hamiltonian Cycle and Edge Dominating Set algorithms with running times on the form $n^{f(\mathbf{cwd}(G))}$ were derived, but no FPT algorithms for Hamiltonian Cycle and Edge Dominating Set were found. Another classical problem that is known to be FPT parameterized by treewidth and solvable in $n^{f(\mathbf{cwd}(G))}$ time is Graph Coloring. The question on the existence of fixed parameter tractable algorithms (with clique-width being the parameter) for all these problems (or their generalizations) was asked by Gerber and Kobler [73], Kobler and Rotics [96, 97], Makowsky, Rotics, Averbouch, Kotek, and Godlin [105, 74]. In the next sections we show that Graph Coloring, Hamiltonian Cycle and Edge Dominating Set all are W[1]-hard parameterized by the cliquewidth of the input graph. This implies that unless FPT $=$ W[1], any extension of Courcelle's theorem to graphs of bounded cliquewidth can not apply to all problems expressible in MSO_2.

7.2.1 Graph Coloring — Chromatic Number

In this section, we prove that Graph Coloring is $W[1]$-hard parameterized by clique-width.

> Graph Coloring (or Chromatic Number): The chromatic number of a graph $G = (V(G), E(G))$ is the smallest number of colors $\chi(G)$ needed to color the vertices of G so that no two adjacent vertices are of the same color.

Our reduction is from the Equitable Coloring problem parameterized by the number r of colors used, and the treewidth of the input graph. In the

EQUITABLE COLORING problem one is given a graph G and integer r and asked whether G can be properly r-colored in such a way that the number of vertices in any two color classes differs by at most 1. Notice that if n is divisible by r this implies that all color classes must contain the same number of vertices. In our reduction we will assume that in the instance we reduce from, n is divisible by r. For a justification of this assumption, if r does not divide n we can add a clique of size $n+r-\lfloor \frac{n}{r} \rfloor r$ to G. We reduce from the exact version of EQUITABLE COLORING, that is, the version where we are looking for an equitable coloring of G with exactly r colors. Recall that in Section 7.1.1 we proved that EQUITABLE COLORING is $W[1]$-hard parameterized by the treewidth t of the input graph and the number of colors r.

Construction: On input (G, r) to EQUITABLE COLORING, we construct an instance (G', r') of GRAPH COLORING as follows. We start with a copy of G and let $r' = r + nr$. We now add a clique P of size r' to G'. The clique P will function as a *palette* in our reduction, as we have to use all r' available colors to properly color it. We partition P into $r+1$ parts as follows, $P = P^M \cup P_1 \cup P_2 \cdots \cup P_r$ where P^M has size r and P_i has size n for every i. We call P^M the main palette, and denote the vertices in P^M by p_i for $1 \le i \le r$. We add edges between every vertex of $P \setminus P^M$ and every vertex of the copy of G. For each vertex $u \in V(G)$ we assign a vertex $u_{P_i} \in P_i$ for every i. Now, for every $1 \le i \le r$ we add a set S_i of vertices. For each vertex $u \in V(G)$ we make a vertex u_{S_i} in S_i for every $1 \le i \le r$, and make u_{S_i} adjacent to u and the entire palette P *except for* u_{P_i} and p_i. We conclude the construction by adding a clique C_i of $n\frac{r-1}{r}$ vertices and making every vertex of C_i adjacent to all of the vertices of S_i and the entire palette except for P_i.

Lemma 7.2.1 *If G has an equitable r-coloring ψ, then G' has an r'-coloring ϕ.*

Proof. We construct a coloring ϕ of G' as follows. The coloring ϕ colors the copy of G in G' in the same way that ψ colors G. We color the palette, assigning a unique color to each vertex and making sure that the main palette P^M is colored using the same colors that are used to color the vertices of G. For every vertex u_{S_i} we color u_{S_i} with $\phi(p_i)$ if $\phi(u) \neq \phi(p_i)$ and with $\phi(u_{P_i})$ if $\phi(u) = \phi(p_i)$. We color every vertex of C_i with some color from P_i (a color used to color a vertex of P_i). To do this we need $n\frac{r-1}{r}$ different colors from P_i. Since exactly n/r vertices of G are colored with $\phi(p_i)$, exactly $n\frac{r-1}{r}$ of S_i are colored with $\phi(p_i)$ and thus n/r vertices of S_i are colored with colors of P_i. Hence there are $n\frac{r-1}{r}$ colors of P_i available to color C_i. Thus, ϕ is a proper r'-coloring of G concluding the proof. ∎

Lemma 7.2.2 *If G' has an r'-coloring ϕ, then G has an equitable r-coloring ψ.*

Proof. We prove that the restriction of ϕ to the copy of G in G' in fact is an equitable r-coloring of G. Since ϕ can only use the colors of P^M, ϕ is a proper r-coloring of G. It remains to prove that for any i between 1 and r, at most n/r vertices of G are colored with $\phi(p_i)$. Suppose for contradiction that there is an i such that more than n/r vertices of G are colored with $\phi(p_i)$. Then there are more than n/r vertices of S_i that are colored with colors of P_i. Since each such vertex must take a different color from P_i, there are less than $n\frac{r-1}{r}$ different colors of P_i available to color the vertices of C_i. However, since C_i is a clique on $n\frac{r-1}{r}$ vertices that must be colored with colors of P_i, this is a contradiction. ∎

Lemma 7.2.3 *If the treewidth of G is t, then the cliquewidth of G' is at most $k = 3 \cdot 2^{t-1} + 7r + 3$. Furthermore, an expression tree of width k for G' can be computed in FPT time.*

Proof. By Theorem 1.2.1, we can compute an expression tree for G of width at most $3 \cdot 2^{t-1}$ in FPT time. Our strategy is as follows. We first show how to modify the expression tree to give a width k expression tree for $G' \setminus (P^M \bigcup_{i=1}^{r} C_i)$. Then we change this tree into an expression tree for G'. In order to give an expression tree for G' we introduce the following extra labels.

- For every $1 \leq i \leq r$ the labels α_i, α_i^L and α_i^R for vertices in P_i.
- For every $1 \leq i \leq r$ the labels β_i, β_i^L and β_i^R for vertices in S_i.
- For every $1 \leq i \leq r$ the label ζ_i for vertices in C_i.
- A "work" label – γ^W, and a label γ^M for P_M.

In the expression tree for G, we replace every introduce-node $i(v)$ with a small expression tree $T_i(v)$. In $T_i(v)$, the vertex v is introduced with label γ^W and the vertices $v_{P_1}, \ldots, v_{P_r}$ and $v_{S_1}, \ldots, v_{S_r}$ are introduced with labels $\alpha_1, \ldots, \alpha_r$ and $\beta_1, \ldots, \beta_r$ respectively. Also, γ^W is joined to $\beta_1, \ldots, \beta_r$ and for every p, β_p is joined with every label in $\{\alpha_q : q \neq p\}$. Also, for every $p \neq q$, α_p is joined with α_q. Finally, γ^W is relabelled to i.

Now, for every union node in the expression tree (not the union nodes inside the T_i's) we add extra vertices on the edges incident to this node. On the edge from the node to its left child, we add nodes that relabel α_p to α_p^L and β_p to β_p^L for every p. Similarly, on the edge from the union node to its right child, we add nodes that relabel α_p to α_p^R and β_p to β_p^R for every p. Finally, on the edge from the union node to its parent we add nodes that first join every α_p^L with every β_q^R and α_q^R, join every α_p^R with every β_q^L, and then relabel every α_p^L and α_p^R to α_p and every β_p^L and β_p^R to β_p.

To conclude the construction of $G' \setminus (P^M \bigcup_{i=1}^{r} C_i)$ we need to add some extra nodes above the root of the expression tree. We add the edges between $P \setminus P^M$ and G by joining every α_p with all labels used for constructing G.

We now need to add the construction of P^M and $\bigcup_{i=1}^{r} C_i$ to our expression tree. We start by making C_p for every p between 1 and r. For every p we add a clique on $n\frac{r-1}{r}$ vertices labelled ζ_p. Every ζ_p is joined to β_p and for every pair $p \neq q$, ζ_p is joined with α_q.

Finally, we add the construction of P^M. For every i, we introduce the vertex p_i with label γ^W, join γ^W to α_j and ζ_j for every j, γ^W with β_j for every $j \neq i$ and finally join γ^W to γ^M and relabel γ^W to γ^M. This concludes the construction of G'. Notice that this expression tree for G' uses $k = 3 \cdot 2^{t-1} + 9r + 3$ labels. ■

Lemmas 7.2.1, 7.2.2 and 7.2.3 together imply the following result.

Theorem 7.2.4 *The* Graph Coloring *problem is $W[1]$-hard when parameterized by clique-width. Moreover, this problem remains $W[1]$-hard even if the expression tree is given.*

7.2.2 Edge Dominating Set

In this section, we show that Edge Dominating Set is $W[1]$-hard parameterized by clique-width.

> Edge Dominating Set: Given a graph $G = (V, E)$, find a minimum set of edges $X \subseteq E(G)$ such that every edge of G is either included in X, or it is adjacent to at least one edge of X. The set X is called an *edge dominating set* of G

Our reduction is from a variant of Capacitated Dominating Set problem, Exact Saturated Capacitated Dominating Set: A *capacitated graph* is a pair (G, c), where G is a graph and $c \colon V(G) \to \mathbb{N}$ is a *capacity* function such that $1 \leq c(v) \leq \deg(v)$ for every vertex $v \in V(G)$ (sometimes we simply say that G is a capacitated graph if the capacity function is clear from the context). A set $S \subseteq V(G)$ is called a *capacitated dominating set* if there is a *domination mapping* $f \colon V(G) \setminus S \to S$ which maps every vertex in $(V(G) \setminus S)$ to one of its neighbors such that the total number of vertices mapped by f to any vertex $v \in S$ does not exceed its capacity $c(v)$. We say that for a vertex $u \in S$, vertices in the set $f^{-1}(u)$ are *dominated by u*. The Capacitated Dominating Set problem is formulated as follows: given a capacitated graph (G, c) and a positive integer k, determine whether there exists a capacitated dominating set S for G containing at most k vertices. Recall that in Section 7.1.2 we proved this problem is $W[1]$-hard when parameterized by treewidth.

For the intractability proof of EDGE DOMINATING SET, we need a special variant of CAPACITATED DOMINATING SET problem which we call EXACT SATURATED CAPACITATED DOMINATING SET. Given a capacitated dominating set S, a vertex $v \in S$ is called *saturated* if the corresponding domination mapping f maps $c(v)$ vertices to v, that is, $|f^{-1}(v)| = c(v)$. A capacitated dominating set $S \subseteq V(G)$ is called *saturated* if there is a domination mapping f which saturates all vertices of S. In EXACT SATURATED CAPACITATED DOMINATING SET a capacitated graph (G, c) and a positive integer k is given. The question is whether G has a saturated capacitated dominating set S with exactly k vertices.

Lemma 7.2.5 *The* EXACT SATURATED CAPACITATED DOMINATING SET *problem is $W[1]$-hard when parameterized by clique-width. Moreover, this problem remains $W[1]$-hard even if the expression tree is given.*

Proof. We show a stronger statement, namely that the EXACT SATURATED CAPACITATED DOMINATING SET problem is $W[1]$-hard when parameterized by the treewidth of the input graph. The statement of the lemma then follows directly from Theorem 1.2.1. To show that the EXACT SATURATED CAPACITATED DOMINATING SET problem is $W[1]$-hard when parameterized by the treewidth of the input graph, we consider the reduction from MULTICOLOR CLIQUE to CAPACITATED DOMINATING SET presented in Section 7.1.2. We tweak the reduction a little bit by for every $i, j \leq k$ giving all the neighbours of $\hat{x}_i$ and $\hat{x}_{ij}$ capacity *one less* than their degree instead of their degree. Let c' be the tweaked capacity function. One can now verify that (H, c', k') has a capacitated dominating set of size at most k' if and only if (H, c, k') does, and that if (H, c', k') has a capacitated dominating set of size at most k' then this set has size exactly k' and is saturated. Observe also that the reduction from the marked version of CAPACITATED DOMINATING SET to the original version preserves exact saturating capacitated dominating sets.

■

We now proceed to show that EDGE DOMINATING SET is $W[1]$-hard parameterized by clique-width by giving a reduction from EXACT SATURATED DOMINATING SET. We start with descriptions of auxiliary gadgets.

Auxiliary gadgets: Let $s \leq t$ be positive integers. We construct a graph $F_{s,t}$ with the vertex set $\{x_1, \ldots, x_s, y_1, \ldots, y_s, z_1, \ldots, z_t\}$ and edges x_iy_i, $1 \leq i \leq s$ and y_iz_j, $1 \leq i \leq s$ and $1 \leq j \leq t$. Basically we have complete bipartite graph between y_i's and z_j's with pendent vertices attached to y_i's. The vertices $z_1, z_2, \ldots, z_t$ are called *roots* of $F_{s,t}$.

Graph $F_{s,t}$ has the following property.

Lemma 7.2.6 *Any set of s edges incident to vertices $y_1, \ldots, y_s$ forms an edge dominating set in $F_{s,t}$. Furthermore, let G be a graph obtained by the union of*

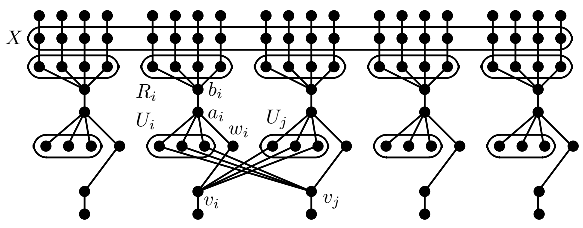

Figure 7.4: Graph G'

$F_{s,t}$ with some other graph H such that $V(F_{s,t}) \cap V(H) = \{z_1, \dots, z_t\}$. Then every edge dominating set of G contains at least s edges from $F_{s,t}$.

The proof of the lemma follows from the fact that every edge dominating set includes at least one edge from $E(y_i)$ for $i \in \{1, \dots, s\}$.

Now we describe our reduction. Let (G, c) be a capacitated graph with the vertex set $\{u_1, \dots, u_n\}$, and k be a positive integer. For every vertex u_i, the set U_i with $c(u_i)$ vertices is introduced, and then vertex sets $\{v_1, \dots, v_n\}$ and $\{w_1, \dots, w_n\}$ are added. For every edge $u_i u_j \in E(G)$, all vertices of U_i are joined with v_j and all vertices of U_j are joined with v_i by edges. Then every vertex v_i is joined to its counterpart w_i and to every vertex v_i we add one additional leaf (a pendent vertex). Now vertex sets $\{a_1, \dots, a_n\}$ and $\{b_1, \dots, b_n\}$ are constructed, and vertices a_i are made adjacent to all vertices of U_i, w_i and b_i. For every vertex b_i, a set R_i of $c(u_i) + 1$ vertices is added and b_i is made adjacent to all the vertices in R_i. Then we add to every vertex of $R_1 \cup R_2 \cup \dots \cup R_n$ a path of length two. Let X be the set of middle vertices of these paths. We denote the obtained graph by G' (see Fig 7.4). Finally, we introduce three copies of $F_{s,t}$:

- a copy of $F_{n-k,n}$ with roots $\{a_1, \dots, a_n\}$,
- a copy of $F_{k,n}$ with roots $\{b_1, \dots, b_n\}$, and a
- a copy of $F_{n,r}$ where $r = \sum_{i=1}^{n} c(u_i)$ with roots in X.

Let this final resulting graph be H.

Lemma 7.2.7 *A graph G has an exact saturated dominating set of the size k if and only if H has an edge dominating set of cardinality at most $2n + r$.*

Proof. Let S be an exact saturated dominating set of the size k in G and f be its corresponding domination mapping. For convenience (without loss of a generality) we assume that $S = \{u_1, \dots, u_k\}$. We construct the edge dominating set as follows. First we select an edge emanating from every vertex in the set $\{v_1, \dots, v_n\}$. For every vertex v_i, $1 \le i \le k$, the edge $v_i w_i$ is selected. Now let us assume that $k < i \le n$ and $f(u_i) = u_j$. We choose a vertex u in U_j which is not incident to already chosen edges and add the edge uv_i to our set. Notice that we always have such a choice of $u \in U_j$ as $c(u_j) = |U_j|$. We observe that these edges already dominates all the edges in the sets $E(v_i)$, $1 \le i \le n$, and in sets $E(u)$ for $u \in U_1 \cup \dots \cup U_k \cup \{w_1, \dots, w_k\}$. Now we add $n-k$ edges from $F_{n-k,n}$ which are incident to vertices in $\{a_{k+1}, \dots, a_n\}$ and k edges from $F_{k,n}$ which are incident to $\{b_1, \dots, b_k\}$. Then $r-n$ matching edges joining vertices of $R_{k+1}, \dots, R_n$ to the vertices of X are included in the set. Finally, we add n edges form $F_{n,r}$ which are incident to vertices of X which are adjacent to vertices of $R_1, \dots, R_k$. Since S is an exact capacitated dominating set, $\sum_{i=1}^{k}(c(u_i)+1) = n$, and from our description it is clear that the resulting set is an edge dominating set of size $2n+r$ for H.

We proceed by proving the other direction of the equivalence. Let L be an edge dominating set of cardinality at most $2n+r$. The set L is forced to contain at least one edge from every $E(v_i)$, at least $n-k$ edges from $F_{n-k,n}$, at least k edges from $F_{k,n}$, and at least one edge from $E(x)$ for all $x \in X$ because of pendent edges. This implies that $|L| = 2n+r$, and L contains exactly one edge from every $E(v_i)$, exactly $n-k$ edges from $F_{n-k,n}$, exactly k edges from $F_{k,n}$, and exactly one edge from $E(x)$ for all $x \in X$. Every edge a_ib_i needs to be dominated by some edge of L, in particular it must be dominated from either an edge of $F_{n-k,n}$, or $F_{k,n}$. Let $I = \{i : a_i$ is incident to an edge from $L \cap E(F_{n-k,n})\}$ and $J = \{j \colon b_j$ is incident to an edge from $L \cap E(F_{k,n})\}$. The above constraints on the set L implies that $|I| = n-k$, $|J| = k$, and these sets form a partition of $\{1, \dots, n\}$. The edges which join vertices b_i and R_i for $i \in I$ are not dominated by edges from $L \cap E(F_{k,n})$. Hence to dominate these edges we need at least $\sum_{i \in I} |R_i|$ edges which connect sets R_i and X. Since at least n edges of $F_{n,r}$ are included in L, we have that $\sum_{i \in I} |R_i| \le r-n$ and $\sum_{j \in J} |R_j| = r - \sum_{i \in I} |R_i| \ge r-(r-n) \ge n$. Let $S = \{u_j \colon j \in J\}$. Clearly, $|S| = k$. Now we show that S is a saturated capacitated dominating set. For $j \in J$, edges which join a vertex a_j to U_j and w_j are not dominated by edges from $L \cap E(F_{n-k,k})$, and hence they have to be dominated by edges from sets $E(v_i)$. Since $n \le \sum_{j \in J} |R_j| = \sum_{j \in J}(|U_j|+1)$, there are exactly n such edges, and every such edge must be dominated by exactly one edge from L. An edge a_jw_j can only be dominated by edge v_jw_j. We also know that $L \cap E(v_i) \ne \emptyset$ for all $i \in \{1, \dots, n\}$ and hence for every v_i, $i \notin J$, there is exactly one edge which joins it with some vertex $u \in U_j$ for some $j \in J$. Furthermore, all these edges are not adjacent, that is, they form a matching. We define $f(u_i) = u_j$ for $i \notin J$. From our construction it follows that f is a domination mapping for S and S is an exact saturated dominating set in G. ∎

The next lemma shows that if the graph G we started with has bounded clique-width then H also has bounded clique-width.

Lemma 7.2.8 *If* $\mathbf{cwd}(G) \le t$ *then* $\mathbf{cwd}(H) \le 2t+16$*, and an expression tree* *for H of width at most $w = 2t + 16$ can be constructed in a polynomial time from the expression tree for G.*

Proof. The graph G is of clique-width at most t. Suppose that the expression tree for G uses t-labels $\{\alpha_1, \ldots, \alpha_t\}$. To construct the expression tree for H we need following additional labels:

- Labels $\beta_1, \ldots, \beta_t$ for the vertices in $U_1, \ldots, U_n$.
- Labels ξ_1, ξ_2, and ξ_3 for attaching $F_{n-k,n}$, $F_{k,n}$ and $F_{n,r}$ respectively.
- Labels $\zeta_1, \ldots, \zeta_4$ for marking some vertices like $w_1, \ldots, w_n$.
- Working labels $\gamma_1, \ldots, \gamma_9$.

When a vertex $u_i \in V(G)$ labeled α_j is introduced, we perform following set of operations. First we introduce following vertices with some working labels: v_i with label γ_1, $c(u_i)$ vertices of U_i with label γ_2, the vertex w_i with label γ_3, and the additional vertex (the leaf attached to v_i) with label γ_4. Now we join the vertex labelled with γ_1 to vertices labelled with γ_3 and γ_4 (basically joining v_i with w_i and its pendent leaf). Finally, we relabel γ_4 to ζ_1 and γ_1 to β_j. Now we introduce vertices a_i and b_i with labels γ_5 and γ_6 respectively. Then we join the vertex labelled γ_4 (a_i) with all the vertices labelled with γ_2, γ_3 and γ_6 (U_i, w_i, b_i). The join operation is followed by relabeling γ_3 to ζ_2, γ_2 to α_j and γ_5 with ξ_1.

Now we want to make the vertices of R_i and the paths attached to it. To do so we perform following operations $c(u_i) + 1$ times: (a) introduce three nodes labelled with γ_7, γ_8 and γ_9 (b) join γ_6 with γ_7, γ_7 with γ_8 and γ_8 with γ_9 and (c) finally we relabel γ_6 to ξ_2, γ_7 to ζ_3, γ_8 to ξ_3 and γ_9 to ζ_4. We omit the union operations from the description and assume that if some vertex is introduced then this operation is performed.

If in the expression tree of G, we have join operation between two labels say α_i and α_j then we simulate this by applying join operations between α_i and β_j and α_j and β_i. The relabel operation in the expression tree of G, that is, relabel α_i to α_j is replaced by relabel α_i to α_j and relabel β_i to β_j. Union operations in the expression tree is done as before.

Finally to complete the expression tree for H, we need to add $F_{n-k,n}$, $F_{k,n}$ and $F_{n,r}$. Notice that all the vertices in $\{a_1, \ldots, a_n\}$, $\{b_1, \ldots, b_n\}$ and X are labelled ξ_1, ξ_2 and ξ_3 respectively. From here we can easily add $F_{n-k,n}$, $F_{k,n}$ and $F_{n,r}$ with root vertices $\{a_1, \ldots, a_n\}$, $\{b_1, \ldots, b_n\}$ and X respectively by using working labels. This concludes the description for the expression tree for H. ∎

Lemmas 7.2.7 and 7.2.8 together imply the following result.

Theorem 7.2.9 *The* Edge Dominating Set *problem is $W[1]$-hard when parameterized by clique-width. Moreover, the problem remains $W[1]$-hard even if the expression tree is given.*

7.2.3 Hamiltonian Cycle

In this section we show that the Hamiltonian Cycle problem is $W[1]$-hard for the graphs of bounded clique-width.

> Hamiltonian Cycle: Given a graph G, check whether there exists a cycle passing through every vertex of G.

Our reduction is from the Capacitated Dominating Set problem described in Section 7.1.2 and shown to be $W[1]$-hard in Theorem 7.1.9. We describe a few auxiliary gadgets.

Auxiliary gadgets: We denote by L_1, the graph with the vertex set $\{x, y, z, a, b, c, d\}$ and the edge set $\{xa, ab, bc, cd, dy, bz, cz\}$. Let P_1 be the path $xabzcdy$, and $P_2 = xabcdy$. (See Fig. 7.5.)

We abstract a property of this graph in the following lemma.

Lemma 7.2.10 *Let G be a Hamiltonian graph such that $G[V']$ is isomorphic to L_1. Furthermore, if all edges in $E(G) \setminus E(G[V'])$ incident to V' are incident to the copies of the vertices x, y, and z in V', then every Hamiltonian cycle in G either includes the path P_1, or the path P_2 as a segment.*

Our second auxiliary gadget is the graph L_2. This graph has $\{x, y, z, s, t, a, b, c, d, e, f, g, h\}$ as its vertex set. We first include following $\{xa, ab, bz, cz, cd, dy, se, ef, fb, ch, hg, gt\}$ in its edge set. Then x, y-path of length 10 $xw_1 \cdots w_9 y$ is added, and edges $fw_3, w_1w_6, w_4w_9, w_7h$ are included in the set of edges. Let $P = xabzcdy$, $R_1 = sefbaxw_1w_2 \ldots w_9ydchgt$, and $R_2 = sefw_3w_2w_1w_6w_5w_4w_9w_8w_7hgt$. (See Fig. 7.5.) This graph has the following property.

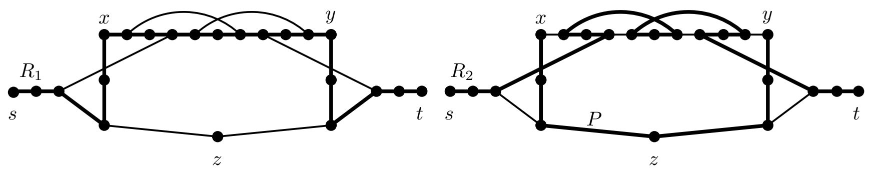

Figure 7.5: Graphs L_1 and L_2. Paths P_1, P_2, R_1, R_2 and P are shown by thick lines

Lemma 7.2.11 *Let G be a Hamiltonian graph such that $G[V']$ is isomorphic to L_2. Furthermore, if all edges in $E(G) \setminus E(G[V'])$ incident to V' are incident to the copies of the vertices x, y, z, s, t in V', then every Hamiltonian cycle in G includes either the path R_1, or two paths P and R_2 as segments.*

The lemma easily follows from the presence of degree 2 vertices in the graph L_2, since for any such a vertex, it and adjacent vertices have to belong to one segment of a Hamiltonian path.

Now we are ready to describe our reduction. Let (G, c) be a capacitated graph with the vertex set $\{v_1, \ldots, v_n\}$, m edges, and let k be a positive integer. For every vertex v_i, four vertices a_i, b_i, c_i and w_i are introduced and the vertices b_i and c_i are joined by $c(v_i)+1$ paths of length two. Let C_i denote the set of middle vertices of these paths, and $X_i = C_i \cup \{a_i, b_i, c_i\}$. Then a copy L_2^i of the graph L_2 with $z = w_i$ is added and vertices x and y of this gadget are joined by edges to a_i and b_i respectively. By s_i and t_i we denote the vertices s and t of L_2^i. For every ordered pair $\{v_i, v_j\}$ such that $v_iv_j \in E(G)$, a copy L_2^{ij} of L_2 is attached with $z = w_j$ and vertices x and y made adjacent to all the vertices of C_i. The vertices corresponding to s and t are called s_{ij} and t_{ij} in L_2^{ij}. Furthermore, let x_{ij} and y_{ij} denote the vertices corresponding to x and y in L_2^{ij}. The path corresponding to P in L_2^i is called P^i. Similarly, the path corresponding to P, R_1 and R_2 are called P^{ij}, R_1^{ij} and R_2^{ij} respectively in L_2^{ij}. Denote the obtained graph by $G'(c)$. (See Fig. 7.6 for an illustration.)

In the next step we add two vertices g and h which are joined by $\sum_{i=1}^{n}(c(v_i)+4)+n+2m+1$ paths of length two. Let Y be the set of middle vertices of these paths. All vertices s_i, t_i, s_{ij} and t_{ij} are joined by edges with all vertices of Y. For every vertex r such that $r \in X_i$ (recall $X_i = C_i \cup \{a_i, b_i, c_i\}$), $i \in \{1, \ldots, n\}$, a copy L_1^r of L_1 with $z = r$ is attached and the vertices x, y of this gadget are joined to all vertices of Y. We let x_r and y_r denote the vertices corresponding to x and y in L_1^r. Similarly P_1^r and P_2^r denotes paths in L_1^r corresponding to P_1 and P_2 respectively.

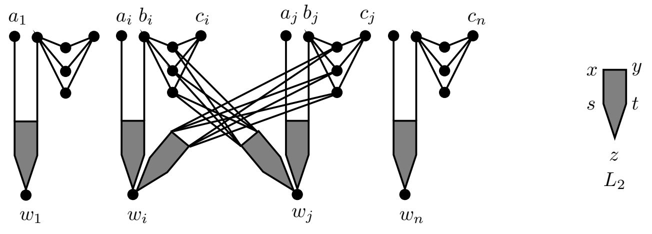

Figure 7.6: Graph $G'(c)$

Finally we add $k+1$ vertices, namely $\{p_1, \ldots, p_{k+1}\}$, and make them adjacent to all the vertices $\{a_i, c_i : 1 \le i \le n\}$ and to g and h. Let this resultant graph be H. The construction of H can easily be done in time polynomial in n and m.

Lemma 7.2.12 *A graph (G, c) has a capacitated dominating set of size at most k if and only if H has a Hamiltonian cycle.*

Proof. Let S be a capacitated dominating set of size at most k in (G, c) with the corresponding dominating mapping f. Without loss of a generality we assume that $|S| = k$ and $S = \{v_1, \ldots, v_k\}$. The Hamiltonian cycle we are trying to construct is naturally divided into $k+1$ parts by the vertices $\{p_1, \ldots, p_{k+1}\}$. We construct the Hamiltonian cycle starting from the vertex p_1. Assume that the part of the cycle up to the vertex p_i is already constructed. We show how to construct the part from p_i to p_{i+1}. We include the edge $p_i a_i$ in it. We add to the cycle the path P^i and two edges, which join the endpoints of P_i with a_i and b_i. Let $J = \{j \colon f(v_j) = v_i\}$. If $J = \emptyset$ then a $b_i - c_i$-path of length two which goes through one vertex of C_i is included in the cycle. Otherwise all paths P^{ij} for $j \in J$ are included in the cycle as follows. We consider the paths P^{ij} in the increasing order of indices in J and add them to the cycle. We take the first path say $P^{ij'}$ and attach $x_{ij'}$ and $y_{ij'}$ to a pair of vertices $\{j_1, j_2\}$ in C_i. Suppose iteratively we have included first $l \geq 1$ paths in J, and the l^{th} path is incident to some $\{j_l, j_{l+1}\}$ in C_i, now we attach the $(l+1)^{th}$ path by attaching x_{it} of this to j_{l+1} and y_{it} of this to j_{l+2}, where j_{l+2} is a new vertex not incident to any previously included paths. We can always find such a vertex as $|J| \leq c(v_i) = |C_i| - 1$. Now we include the edge $b_i j_1$ and $j_{|J|+1} c_i$. Finally we include the edge $c_i p_{i+1}$.

When the vertex p_{k+1} is reached we move to the set Y. Note that at this stage all vertices $\{w_1, \ldots, w_n\}$ are already included in the cycle. We start by including the edge $p_{k+1} g$. We will add following segments to the cycle an connect them appropriately.

- For every L_2^i we add the path R_1^i to the cycle if P^i was not included to it, and include the path R_2^i otherwise. The number of such paths is n.
- Similarly, for every L_2^{ij}, the path R_1^{ij} is added to the cycle if P^{ij} was not included to it, else the path R_2^{ij} is added. Note that $2m$ such paths are included to the cycle.
- For every vertex r such that $r \in X_i$ for some $i \in \{1, \ldots, n\}$, the path P_2^r is included in the cycle if r is already included in the constructed part of the cycle, else the path P_1^r is added. Clearly, we add $\sum_{i=1}^{n} (c(v_i) + 4)$ paths.

Finally the total number of paths we will add is $\sum_{i=1}^{n} (c(v_i)+4) + n + 2m = |Y| - 1$. We add the segments of the paths mentioned with the help of vertices in Y, in the way we added the paths P^{ij} with the help of vertices in C_i. Let the end points of the resultant joined path be $\{q_1, q_2\}$. Notice that (a) $q_1, q_2 \in Y$ and (b) this path include all the vertices of Y. Now we add edges gq_1, $q_2 h$ and hp_1. This completes the construction of the Hamiltonian cycle.

For the reverse direction of the proof, we assume that we have been given C, a Hamiltonian cycle in H. Let $S = \big\{v_i \bigm| p_j a_i \in E(C), a_i p_s \notin E(C), j \neq s$, for some $j \in \{1, 2, \ldots, k+1\}\big\}$. We prove that S is a capacitated dominating set in G of cardinality at most k. We first argue about the size of S, clearly its size is upper bounded by $k+1$. To argue that it is at most k, it is enough to observe that by Lemmas 7.2.10 and 7.2.11 either p_jg, or p_jh must be in $E(C)$ for some $j \in \{1, \ldots, k+1\}$. Now we show that S is indeed a capacitated dominating set. Our proof is based on following observations.

- Every vertex w_j, either appear in a vertex segment, that is P^j, or an edge segment, that is, P^{ij} for some $j \in \{1, \ldots, n\}$ in C.
- If some P^{ij} appear as a segment in C, then from the gadgets $L_1^{b_i}$ and $L_1^{c_i}$ the paths $P_2^{b_i}$ and $P_2^{c_i}$ are part of C. Hence the only way to include b_i in C is by using the edge incident to it from the gadget L_2^i. This implies that from the gadget L_2^i we use the path P^i and two edges, which join the endpoints of P_i with a_i and b_i.
- By Lemma 7.2.10 the cycle contains the edge which joins a_i to some vertex in $\{p_1, \ldots, p_{k+1}\}$.

Now given $v_j \in V(G) \setminus S$, for the domination function f, we assign it to v_i for which P^{ij} is segment in C. Clearly $v_i \in S$ as by above observation there exits a $j \in \{1, 2, \ldots, k+1\}$ such that $p_ja_i \in E(C)$, $a_ip_s \notin E(C)$ and $j \neq s$. For every $v_i \in S$, the set $f^{-1}(v_i)$ contains at most $c(v_i)$ vertices as $|C_i| = c(v_i) + 1$. This concludes the proof. ■

The next lemma upper bounds the clique-width of the resulting graph H.

Lemma 7.2.13 *If* $\mathbf{tw}(G) \leq t$ *then* $\mathbf{cwd}(H) \leq 9 \cdot 2^{\max\{2t,24\}} + 12$ *and an expression tree for H of width at most $w = 9 \cdot 2^{\max\{2t,24\}} + 12$ can be constructed in FPT time.*

Proof. We define $c'(v_i) = 0$ for all $i \in \{1, 2, \ldots, n\}$ and consider the graph $G'(c')$. It is easy to see that $\mathbf{tw}(G'(c')) \leq \max\{2t+1, |V(L_2)| + 3\} = \max\{2t+1, 25\}$. By Theorem 1.2.1 $\mathbf{cwd}(G'(c')) \leq 3 \cdot 2^{\max\{2t,24\}}$, i.e. we can construct the labeled graph $G'(c')$ by using at most $l = 3 \cdot 2^{\max\{2t,24\}}$ labels $\alpha_1, \ldots, \alpha_l$. Using $l+1$ additional labels $\beta_1, \ldots, \beta_l$ and γ_1 we can ensure that all vertices s_i, t_i, s_{ij} and t_{ij} are labeled by the label γ_1, and only these vertices have label γ_1 in the following way. At the moment when such a vertex r labeled e.g. j is introduced, we label it by the label β_j, and then these labels are used in the operations in same way as labels α_j. Finally, all vertices labeled by these labels are relabeled γ_1. Similarly, by using $l+1$ more labels we assume that all vertices a_i and c_i are labeled by the label γ_2, and this label is used only for these vertices. Denote by d_i the only vertex in the set C_i in $G'(c')$. The graph $G'(c)$ can be obtained from $G'(c')$ by the substitution of d_i by $c(v_i) + 1$ vertices with same neighborhoods. This operation does not change clique-width, and $\mathbf{cwd}(G'(c)) \leq 3l + 2$.

Recall that for every vertex $r \in X_i$ we add a copy of L_1 with $z = r$. We show how to construct the obtained graph using no more than $|V(L_1)| + 1 = 8$

additional labels, in such a way that vertices x_r and y_r are labeled by the label γ_1. When a vertex r is introduced, we construct a copy of L_1 using $|V(L_1)|$ extra labels making sure that the z in this copy gets r's label. Then we relabel x and y by γ_1, and the remaining $|V(L_1)| - 3$ vertices are relabeled by an additional label, ζ, which acts as a "waste" label. We use 2 labels to construct the vertices g and h with $|Y|$ paths of length two between them. Additionaly, we ensure that at the end of this construction g and h are labeled with γ_2, and that the vertices of Y are labeled by γ_3.

Now, the join operation is done for vertices labeled γ_1 and γ_3. Now by using one more label γ_4, the vertices $p_1, p_2, \ldots, p_{k+1}$ are introduced, and the join operation is performed on the labels γ_2 and γ_4. We used no more than $3l + 12$ labels to construct H, and $\mathbf{cwd}(H) \leq 3l + 12 \leq 9 \cdot 2^{\max\{2t,24\}} + 12$. The second claim of the lemma is obvious. ∎

Lemmas 7.2.12 and 7.2.13 together imply the following result.

Theorem 7.2.14 *The* Hamiltonian Cycle *problem is $W[1]$-hard when parameterized by clique-width. Moreover, this problem remains $W[1]$-hard even if the expression tree is given.*

7.2.4 Conclusion

Our results rule out FPT algorithms for Graph Coloring, Hamiltonian Cycle and Edge Dominating Set parameterized by cliquewidth unless FPT=W[1]. We conclude the chapter with two open problems. The best known algorithms for these problems have running times on the form $n^{2^{O(\mathbf{cwd}(G))}}$, an interesting question is, can they be solved in $n^{k^{O(1)}}$ time? Another graph problem expressible in MSO_2 but not MSO_1 is the Max Cut problem. Is this problem also W[1]-hard parameterized by cliquewidth, or can an FPT algorithm be derived for it?

Chapter 8

Kernelization Techniques

8.1 Meta-Theorems for Kernelization

One of the most important results in the area of kernelization is given by Alber et al. [5]. They provided the first kernel of linear size for the Dominating Set problem on planar graphs. The work of Alber et al. [5] has triggered an explosion of papers on kernelization, and in particular on kernelization of problems on planar graphs. Combining the ideas of Alber et al. [5] with problem specific data reduction rules, kernels of linear sizes were obtained for a variety of parameterized problems on planar graphs including Connected Vertex Cover, Minimum Edge Dominating Set, Maximum Triangle Packing, Efficient Edge Dominating Set, Induced Matching, Full-Degree Spanning Tree, Feedback Vertex Set, Cycle Packing, and Connected Dominating Set [5, 22, 23, 28, 79, 80, 87, 104, 112]

Most of the papers on linear kernels on planar graphs have the following idea in common: find an appropriate region decomposition of the input planar graph based on the problem in question, and then perform *problem specific* rules to reduce the part of the graph inside each region. The first step towards the general abstraction of all these algorithms was initiated by Guo and Niedermeier [79], who introduced the notion of "problems with distance property". However, to prove that some problem admits a linear kernel on planar graphs, the approach of Guo and Niedermeier still requires the design of problem specific reduction rules. In this section we show that if a problem satisfies certain conditions, like *expressibility in a certain kind of logic*, then these reduction rules can be automatically generated. We also extend the decomposition theorems proved by Guo and Niedermeier [79] to graphs of bounded genus, and show that our reduction rules apply in bounded genus graphs as well. Our theorems unify and extend *all* previously known kernelization results for planar graph problems.

We say that a parameterized problem $\Pi \subseteq \mathcal{G}_g \times \mathbb{N}$ is *compact* if there exists an integer r such that for all $(G = (V, E), k) \in \Pi$ there is an embedding of G

into a surface Σ of Euler-genus at most g and a set $S \subseteq V$ such that $|S| \leq r \cdot k$, $\mathbf{R}^r_G(S) = V$, and $k \leq |V|^r$. Similarly, Π is *quasi-compact* if there exists an integer r such that for every $(G, k) \in \Pi$ there is an embedding of G into a surface Σ of Euler-genus at most g and a set $S \subseteq V$ such that $|S| \leq r \cdot k$, $\mathbf{tw}(G \setminus \mathbf{R}^r_G(S)) \leq r$ and $k \leq |V|^r$. Notice that if a problem is compact then it is also quasi-compact.

The first result proved in this section concerns a parameterized analogue of graph optimization problems where the objective is to find a maximum or minimum sized vertex or edge set satisfying a CMSO-expressible property. In particular, the problems considered are defined as follows. In a p-MIN-CMSO graph problem $\Pi \subseteq \mathcal{G}_g \times \mathbb{N}$, we are given a graph $G = (V, E)$ and an integer k as input. The objective is to decide whether there is a vertex/edge set S of size at most k such that the CMSO-expressible predicate $P_\Pi(G, S)$ is satisfied. In a p-EQ-CMSO problem the size of S is required to be exactly k and in a p-MAX-CMSO problem the size of S is required to be at least k. The *annotated* version Π^α of a p-MIN/EQ/MAX-CMSO problem Π is defined as follows. The input is a triple $(G = (V, E), Y, k)$ where G is a graph, $Y \subseteq V$ is a set of black vertices, and k is a non-negative integer. In the *annotated version* of a p-MIN/EQ-CMSO graph problem, S is additionally required to be a subset of Y. For the annotated version of a p-MAX-CMSO graph problem S is not required to be a subset of Y, but instead of $|S| \geq k$ we demand that $|S \cap Y| \geq k$.

Our results. For a parameterized problem $\Pi \subseteq \mathcal{G}_g \times \mathbb{N}$, let $\overline{\Pi} \subseteq \mathcal{G}_g \times \mathbb{N}$ denote the set of all *no*-instances of Π. The first main result of this chapter is the following theorem.

Theorem 8.1.1 *Let* $\Pi \subseteq \mathcal{G}_g \times \mathbb{N}$ *be a* p-MIN/EQ/MAX-CMSO *problem and either* Π *or* $\overline{\Pi}$ *is compact. Then the annotated version* Π^α *admits a cubic kernel if* Π *is a* p-EQ-CMSO *problem and a quadratic kernel if* Π *is a* p-MIN/MAX-CMSO *problem.*

We remark that a polynomial kernel for an annotated graph problem Π^α, is a polynomial time algorithm that given an input $(G = (V, E), Y, k)$ of Π^α, computes an equivalent instance $(G' = (V', E'), Y', k')$ of Π^α such that $|V'|$ and $k' \leq k^{O(1)}$. Theorem 8.1.1 has the following corollary.

Corollary 8.1.2 *Let* $\Pi \subseteq \mathcal{G}_g \times \mathbb{N}$ *be an* NP*-complete* p-MIN/EQ/MAX-CMSO *problem such that either* Π *or* $\overline{\Pi}$ *is compact and* Π^α *is in* NP . *Then* Π *admits a polynomial kernel.*

Theorem 8.1.1 and its corollary give polynomial kernels for *all* graph problems on surfaces for which such kernels are known. However, many problems in the literature are known to admit linear kernels on planar graphs. Our next theorem unifies and generalizes *all* known linear kernels for graph problems on surfaces. To this end we utilize the notion of *finite integer index*.

Theorem 8.1.3 *Let* $\Pi \subseteq \mathcal{G}_g \times \mathbb{N}$ *has finite integer index and either* Π *or* $\overline{\Pi}$ *is quasi-compact. Then* Π *admits a linear kernel.*

Theorems 8.1.1 and 8.1.3 will be proved in Section 8.1.3. The theorems are similar in spirit, yet they have a few differences. In particular, not every p-MIN/EQ/MAX-CMSO graph problem has finite integer index. For an example the INDEPENDENT DOMINATING SET problem is a p-MIN-CMSO problem, but it does not have finite integer index. Also, the class of problems that have finite integer index does not have a syntactic characterization and hence it takes some more work to apply Theorem 8.1.3 than Theorem 8.1.1. On the other hand, Theorem 8.1.3 yields linear kernels, applies to quasi-compact problems and is not harder to apply than to design dynamic programming algorithms for graphs of bounded treewidth.

To demonstrate the applicability of our results we show in Section 8.1.4 how our theorems lead to polynomial or linear kernels for a variety of problems. For ease of reference we provide a compendium of parameterized problems for which polynomial or linear kernels are derived in Section 8.1.5. We now proceed to develop the tools necessary to prove Theorems 8.1.1 and 8.1.3.

8.1.1 Reduction Rules

In this section we give reduction rules for compact annotated p-MIN/EQ/MAX-CMSO graph problems and quasi-compact parameterized problems having finite integer index. Our reduction rules have the following form:

> *If there is a constant size separator such that after its removal we obtain a connected component of unbounded size and of constant treewidth, then we replace this component with something of constant size.*

The implementation of this rule depends on whether we are dealing with an annotated p-MIN-CMSO, p-EQ-CMSO or p-MAX-CMSO problems, or whether the problem in question has finite integer index. Our reduction rules for annotated p-MIN/EQ/MAX-CMSO problems have three parts. In the first two parts we zero in on an area to reduce, in the last part we perform the reduction. We now define the notion of *protrusions* which formalizes the notion if a constant size separator whose removal creates a large connected component of constant treewidth.

Definition 8.1.4 [r**-protrusion**] *Given a graph* $G = (V, E)$*, we say that a set* $X' \subseteq V$ *is an* r-protrusion *of* G *if* $|N(X')| \leq r$ *and* $\mathbf{tw}(G[X' \cup N(X')]) \leq r$*. For an* r*-protrusion* X'*, the vertex set* $X = X' \cup N(X')$ *is an* extended r-protrusion. *The set* X *is the extended protrusion of* X' *and* X' *is the protrusion of* X*.*

In the following discussion we only treat annotated p-MIN/EQ/MAX-CMSO problems where the set S being searched for is a set of vertices. The case where S is a set of edges can be dealt in an identical manner.

Reduction for Annotated p-MIN-CMSO Problems

We now describe the reduction rule that we apply for annotated p-MIN-CMSO problems. The technique employed in this section will act as a template for how we handle the annotated p-EQ/MAX-CMSO problems. Recall that in an annotated p-MIN-CMSO problem Π^α we are given a graph $G = (V, E)$ where a subset Y of the vertices of G is colored *black* and an integer k. The objective is to find a set $S \subseteq Y$ of size at most k such that a fixed CMSO-definable property $P_\Pi(G, S)$ holds. For our reduction rule, we are also given a sufficiently large r-protrusion X'. In the first step of the reduction, we show that the set $Y \cap X'$ can be reduced to $O(k)$ vertices without changing whether (G, k) is a *yes*-instance to Π^α or not. In the second step we show that the r-protrusion X' can be covered by $O(k)$ r'-protrusions such that each r'-protrusion contains at most a constant number of vertices from Y. In the third and final step of the reduction rule, we replace the largest r'-protrusion with an equivalent, but smaller r'-boundaried graph. We now provide the reduction rule for annotated p-MIN-CMSO problems.

Lemma 8.1.5 *Let Π^α be an annotated p-MIN-CMSO problem. Let $G = (V, E)$ be a graph, $Y \subseteq V$ be the set of black vertices and k be an integer. Let X be an extended r-protrusion of G. Then there is an integer c, and an $O(|X|)$ time algorithm, that computes a set of vertices $Z \subseteq X \cap Y$ with $|Z| \leq ck$, such that if there exists a $W \subseteq Y$ with $P_\Pi(G, W)$ and $|W| \leq k$, then there exists a $W' \subseteq Y$ with $P_\Pi(G, W')$, $|W'| \leq k'$, and $W' \cap X \subseteq Z$.*

Proof. The algorithm starts by making a tree decomposition of $G[X]$ of width at most r, using the linear time algorithm to compute treewidth by Bodlaender [18]. Now we add $\partial(X)$ to each bag, and add one bag containing only the vertices in $\partial(X)$. The tree decomposition has width at most $2r$ as the bag size is at most $r + 1 + |\partial(X)| \leq 2r + 1$.

Consider the following equivalence relation on subsets $Q \subseteq X \cap Y$. We say that $Q \sim Q'$, if and only if for all $R \subseteq V - X$:

$$P_\Pi(G, Q \cup R) \Leftrightarrow P_\Pi(G, Q' \cup R). \tag{8.1}$$

The number of equivalence classes is bounded by a function of the treewidth [27, 39] of $G[X]$, and thus for fixed r, can be assumed to be bounded by a constant, say c. We would like to find a minimum sized representative of each of the equivalence classes. We describe an algorithm running in time $O(|X|)$ to find the desired set in each equivalence class. Let us consider an algorithm that solves an optimization version of the problem

$$\min\{|W| \mid W \subseteq Y \wedge P_\Pi(G, W)\}$$

on graphs of bounded treewidth, using a dynamic programming approach. For an example, we can use the algorithm described by Borie et al. [27]. The algorithm of

Borie et al. [27] computes for each equivalence class in Relation 8.1 (or, possibly, a refinement of the Relation 8.1) the minimum size of a set in the class. This is done in a dynamic programming fashion, computing each value given a table of these values for the children of the bag in the tree decomposition. The running time is linear for fixed c. It is not hard to observe that we can also compute for each equivalence class a minimum size set $Q \subseteq X \cap Y$ that belongs to the class. This can be done in linear time. If there are more than one minimum size sets in a class, then the algorithm just picks one.

Let $\mathcal{Q}$ denote that set of equivalence classes of Relation 8.1 and suppose that for each class $q \in \mathcal{Q}$ that is non-empty, we have a minimum size representative Q_q. By the above argument we can find a minimum size representative Q_q for each class in $O(|X|)$ time. Now, set

$$Z = \bigcup_{q \in \mathcal{Q}, |Q_q| \leq k} Q_q \ .$$

Suppose now that there exists $W \subseteq Y$ with $P_\Pi(G, W)$ and $|W| \leq k$. Consider the equivalence class q that contains $X \cap W$. Let Q_q be the selected minimum size representative of q. Consider the set $W' = (W \setminus X) \cup Q_q$. As $P_\Pi(G, (W \setminus X) \cup (X \cap W))$, we have that $P_\Pi(G, (W \setminus X) \cup Q_c) = P_\Pi(G, W')$. Since, Q_q is a minimum size representative from q, we have that $|Q_q| \leq |W \setminus X|$, and that $|W'| \leq |W|$. Finally, since the number of equivalence classes in $\mathcal{Q}$ is a function of r and each representative Q_q has size at most k, we have that $|Z| = O(k)$. This proves the lemma. ∎

Using Lemma 8.1.5, we change the set Y to $(Y \setminus X) \cup Z$. We now show how to exploit the fact that Z contains $O(k)$ vertices.

Partitioning Protrusions: In the second step of the reduction rule we partition the extended r-protrusion X into smaller r'-protrusions.

Lemma 8.1.6 *Let $G = (V, E)$ be a graph, $Y \subseteq V$ be the set of black vertices of G and k be an integer. Furthermore, let X be an extended r-protrusion and $Z = X \cap Y$. There is an $O(|X|)$ time algorithm that finds $O(|Z|)$ extended r'-protrusions $X_1, X_2, \ldots, X_\ell$ such that $X = X_1 \cup X_2 \cup \ldots \cup X_\ell$ and for every $i \leq \ell$, $Z \cap X_i \subseteq \partial(X_i)$.*

Proof. We start by making a nice tree decomposition of $G[X]$, and adding $\partial(X)$ to all bags. Now, we mark a number of bags. First, we mark the root bag and each forget node where a vertex in Z is forgotten. As each vertex is forgotten at most once in a nice tree decomposition, so far we have $O(|Z|)$ marked bags. Now, mark each bag that is the lowest common ancestor of two marked bags, until we cannot marks in this fashion. Standard counting arguments for trees show that this operation at most doubles the number of marks. Hence, there are at most $O(|Z|)$ marked bags.

We now split X into $X_1, X_2, \ldots, X_\ell$ as follows: we take parts of the tree decomposition, with internally no marked bags, plus the marked bags at the border. Note that each such part has at most two marked bags, each of size at most $2r$ and thus the size of $\partial(X_i)$ is at most $4r$ for every $i \leq \ell$. Note that by the way we constructed the sets X_i, $Z \cap X_i \subseteq \partial(X_i)$ for every i. ∎

Reducing Protrusions: In the third phase of our reduction rule, we find a protrusion to replace, and perform the replacement. Notice that this part of the reduction works both for annotated p-MIN-CMSO and for annotated p-EQ-CMSO problems.

Lemma 8.1.7 *Let Π^α be an annotated p-*MIN*-*CMSO *or p-*EQ*-*CMSO *problem. There is a fixed constant c depending only on Π^α such that there is an algorithm that given a graph $G = (V, E) \in \mathcal{G}_g$, a set $Y \subseteq V$ of black vertices, an integer k and an extended r-protrusion X with $|X| > c$ such that $Y \cap X \subseteq \partial(X)$, runs in time $O(|X|)$, and produces a graph $G^* = (V^*, E^*) \in \mathcal{G}_g$ such that $|V^*| < |V|$ and $(G^*, k) \in \Pi^\alpha$ if and only if $(G, k) \in \Pi^\alpha$.*

Proof. For two t-boundaried graphs G_1 and G_2, we say that they are equivalent with respect to a subset S of $\partial(G_1) = \partial(G_2)$ if for every $G_3 = (V_3, E_3)$ and set $S' \subseteq V_3$ we have that $P_\Pi(G_1 \oplus G_3, S \cup S')$ if and only if $P_\Pi(G_2 \oplus G_3, S \cup S')$. If G_1 and G_2 are equivalent with respect to S we say that $G_1 \sim_S G_2$. The canonical equivalence relation for CMSO properties with free set variables has finite index [27, 39] and hence the number of equivalence classes of $\sim_S$ depends only on t for every fixed https://www.easychair.org/login.cgi?a=a01749b032b2;iid=12978 $S \subseteq \partial(G_1)$. We make a new equivalence relation defined on the set of t-boundaried graphs belonging to a graph class $\mathcal{G}$. Two t-boundaried graphs $G_1, G_2 \in \mathcal{G}_g$ are equivalent if

- for every t-boundaried graph G_3, $G_1 \oplus G_3$ is legal if and only if $G_2 \oplus G_3$ is legal; and
- for every $S \subseteq \partial(G_1) = \partial(G_2)$, $G_1 \sim_S G_2$.

Now, $\sim_S$ has a finite number of equivalence classes for every $S \subseteq \partial(G_1)$ and the class $\mathcal{G}_g$ is characterized by a finite set of forbidden minors. Hence the number of equivalence classes in the equivalence relation defined above is a function of t. Let $\mathcal{S}$ be a set of r-boundaried graphs containing one smallest representative for each equivalence class of the relation above. Let c the size of the largest graph in $\mathcal{S}$. We have that X is a r-boundaried graph with boundary $\partial(X)$. Let G_1 be a graph in $\mathcal{S}$ such that G_1 and $G[X]$ are equivalent.

Now we replace $G[X]$ with $G_1 = (V_1, E_1)$ in the graph G, and let the resulting graph be $G^* = (V^*, E^*)$. Let Y_1 be the set of black vertices in $\partial(G_1)$. We let $Y^* = Y \setminus X \cup Y_1$ be the set of black vertices in G^*. Since $|X| > c$, we have that $|V_1| < |X|$ and hence $|V^*| < |V|$. It remains to prove that $(G, k) \in \Pi^\alpha$ if and only if $(G^*, k) \in \Pi^\alpha$. In one direction, suppose there is a set $S \subseteq Y$ such that $P_\Pi(G, S)$ holds. Then $S \cap X \subseteq \partial(G[X])$ and since $G[X]$ and G_1 are equivalent with respect to the relation above we have that $P_\Pi(G^*, S)$ holds. In the other direction, suppose $P_\Pi(G^*, S)$ holds. Since $Y^* \cap V_1 \subseteq \partial(G_1)$ and $G[X]$ and G_1 are equivalent with respect to the relation above, we have that $P_\Pi(G, S)$ holds. This concludes the proof. ∎

Lemmata 8.1.5, 8.1.6 and 8.1.7 together yield a reduction rule for all annotated p-MIN-CMSO problems.

Lemma 8.1.8 *Let Π^α be an annotated p-MIN-CMSO problem. There is a fixed constant c depending only on Π^α such that there is an algorithm that given a graph $G = (V, E) \in \mathcal{G}_g$, a set $Y \subseteq V$ of black vertices, an integer k and an extended r-protrusion X with $|X| > ck$, runs in time $O(|X|)$, and produces a graph $G^* = (V^*, E^*) \in \mathcal{G}_g$ such that $|V^*| < |V|$ and $(G^*, k) \in \Pi^\alpha$ if and only if $(G, k) \in \Pi^\alpha$.*

Proof. The algorithm starts by applying Lemma 8.1.5 to X, thus making all but at most ak black vertices uncolored for some fixed constant a. By Lemma 8.1.6, $X = X_1 \cup X_2 \ldots \cup X_{bk}$ for some fixed constant b, where for every i, X_i is an extended $4r$-protrusion such that $Y \cap X_i \subseteq \partial(X_i)$. By the pigeon-hole principle some X_i has size at least $|X|/bk > c/b$. Choose c such that c/b is sufficiently large to apply the algorithm in Lemma 8.1.7, and then apply Lemma 8.1.7 on the extended protrusion X_i. This concludes the proof. ∎

Reduction for Annotated p-EQ-CMSO Problems

In this section we give a reduction rule for annotated p-EQ-CMSO problems. The rule is very similar to the one for the p-MIN-CMSO problems described in the previous section. Therefore we only highlight the differences between the two rules in our arguments. The main difference between the two problem variants is that we need to keep track of solutions of every fixed size between 0 and k, instead of just the smallest one in each class. Because of this we require the protrusion to contain at least ck^2 vertices instead of ck vertices, in order to be able to reduce it.

Lemma 8.1.9 *Let Π^α be an annotated p-EQ-CMSO problem. There is a fixed constant c depending only on Π^α such that there is an algorithm that given a graph $G = (V, E) \in \mathcal{G}_g$, a set $Y \subseteq V$ of black vertices, an integer k and an* *extended r-protrusion X with $|X| > ck^2$, runs in time $O(k|X|)$, and produces a graph $G^* = (V^*, E^*) \in \mathcal{G}_g$ such that $|V^*| < |V|$ and $(G^*, k) \in \Pi^\alpha$ if and only if $(G, k) \in \Pi^\alpha$.*

Proof. We show that if $Y \cap X \geq ak^2$ for some fixed constant k, then we can remove some vertices from Y preserving the answer. The proof proceeds almost as the proof of Lemma 8.1.5. The main difference is that now, instead of taking a minimum size representative from each equivalence class we consider all possible sizes for S between 0 and k, for each equivalence class. That is, for each ℓ, $0 \leq \ell \leq k$, and each equivalence class q, we make a set $Q_{q,\ell} \subseteq X$ from the class with $|Q_{q,\ell}| = \ell$. Now, set

$$Z = \bigcup_{q \in \mathcal{Q}, 0 \leq \ell \leq k, |Q_{q,\ell}| = \ell} Q_{q,\ell} \, .$$

In the dynamic programming algorithm, we must also have one table entry for each class and each size. This gives a running time of $O(k|X|)$ for the first part of the reduction rule.

Next, we remove all vertices in $X \setminus Z$ from Y and apply Lemma 8.1.6. This gives us $X = X_1 \cup X_2 \ldots \cup X_{bk^2}$ for some constant b, where for every i, X_i is an extended $4r$-protrusion with $Z \cap X_i \subseteq \partial(X_i)$.

By the pigeon-hole principle some X_i has size at least $|X|/bk^2 > c/b$. Choose c such that c/b is sufficiently large to apply the algorithm in Lemma 8.1.7, and then we apply Lemma 8.1.7 on the extended protrusion X_i. This concludes the proof. ■

Reduction for Annotated p-MAX-CMSO Problems

We now give a reduction rule for annotated p-MAX-CMSO problems. The rule is still similar to the ones described in the two previous sections, but differs more from the p-MIN-CMSO problems than p-EQ-CMSO did.

Lemma 8.1.10 *Let Π^α be an annotated p-MAX-CMSO problem. There is a fixed constant c depending only on Π^α such that there is an algorithm that given a graph $G = (V, E) \in \mathcal{G}_g$, a set $Y \subseteq V$ of black vertices, an integer k and an extended r-protrusion X with $|X| > ck$, runs in time $O(|X|)$, and produces a graph $G^* = (V^*, E^*) \in \mathcal{G}_g$ such that $|V^*| < |V|$ and $(G^*, k) \in \Pi^\alpha$ if and only if $(G, k) \in \Pi^\alpha$.*

Proof. We again begin in a manner similar to the proof of Lemma 8.1.5. The main ingredient in the proof of Lemma 8.1.5 is that for a given extended r-protrusion X, we consider the equivalence relation $\sim$ on subsets $Q \subseteq X$, where we demand that $Q \sim Q'$ if and only if for all $R \subseteq V - X$: $P_\Pi(Q \cup R) \Leftrightarrow P_\Pi(Q' \cup$

R). The number of equivalence classes of $\sim$ is bounded, but for maximization problems we need to keep the *largest* representative of each class. Hence we can not guarantee that the union of all the representatives we store is bounded by a function of k. To overcome this difficulty we use the expressive power provided by annotation. We compute a largest representative $Q_q \subseteq X$ for each equivalence class q of $\sim$. Then, we build a vertex set $Z \subseteq X \cap Y$ as follows. For each non-empty equivalence class q, if $|Q_q \cap Y| \leq k$, then we add $Q_q \cap Y$ to Z. If $|Q_q \cap Y| > k$, then we select arbitrarily a k-sized subset of $Q_q \cap Y$ and add it to Z. Thus $|Z| \leq ak$ for some constant a. We remove all vertices in $X \setminus Z$ from Y, without changing the membership of (G, k) in Π^α.

Next we apply Lemma 8.1.6 on X. This gives us $X = X_1 \cup X_2 \ldots \cup X_{bk}$ for some constant b, where for every i, X_i is an extended $4r$-protrusion with $Z \cap X_i \subseteq \partial(X_i)$.

We now describe how to modify Lemma 8.1.7 so that it can also be applied to annotated p-MAX-CMSO problems. For a set $S \subseteq Y$ we define $P'_\Pi(G, S)$ as there exists a set S' containing S such that $P_\Pi(G, S')$ holds. Clearly $(G, k) \in \Pi^\alpha$ if and only if there is a set $S \subseteq Y$ of size k such that $P'_\Pi(G, S)$. If the relation $\sim_S$ in Lemma 8.1.7 is defined using $P'_\Pi(G, S)$ instead of $P_\Pi(G, S)$, the proof goes through also for annotated p-MAX-CMSO problems.

By the pigeon-hole principle some X_i has size at least $|X|/bk > c/b$. Choose c such that c/b is sufficiently large to apply the algorithm in the modified version of Lemma 8.1.7, and then apply the modified version of Lemma 8.1.7 to p-MAX-CMSO problems on the extended protrusion X_i. This concludes the proof. ∎

Reductions Based on Finite Integer Index

In the previous sections we gave reduction rules for annotated p-MIN/EQ/MAX-CMSO problems. These reduction rules, together with the results proved later in this article will give quadratic or cubic kernels for the problems in question. However, for many problems we can in fact show that they admit a linear kernel. In this section we provide reduction rules for graph problems that have finite integer index. These reduction rules will yield linear kernels for the problems they apply to. We are now ready to prove the reduction lemma for problems that have finite integer index.

Lemma 8.1.11 *Let $\Pi \subseteq \mathcal{G}_g \times \mathbb{N}$ be problem with finite integer index in $\mathcal{G}_g$ such that either Π or $\overline{\Pi}$ is quasi-compact. There exists a constant c and an algorithm that given a graph $G = (V, E) \in \mathcal{G}$, an integer k and an extended r-protrusion X in G with $|X| > c$, runs in time $O(|X|)$ and returns a graph $G^* = (V^*, E^*) \in \mathcal{G}_g$ and an integer k^* such that $|V^*| < |V|$, $k^* \leq k$ and $(G^*, k^*) \in \Pi$ if and only if $(G, k) \in \Pi$.*

Proof. Let $\mathcal{S}$ be a set of representatives for (Π, r) and let $c = \max_{Y \in \mathcal{S}} |Y|$. Similarly let $\mathcal{S}'$ be a set of representatives for $(\Pi, 2r)$ and let $c' = \max_{Y \in \mathcal{S}'} |Y|$. If $|X| > 3c'$ we find an extended $2r$-protrusion $X' \subseteq X$ such that $c' < |X'| \leq 3c'$ and work on X' instead of X. This can be done in time $O(|X|)$ since $G[X]$ has treewidth at most r. From now on, we assume that $|X| \leq 3c'$. This initial step is the only step of the algorithm that does not work with constant size structures, and hence the running time of the algorithm is upper bounded by $O(|X|)$. The algorithm proceeds as follows.

Because Π has finite integer index there is a graph $H = (V_H, E_H) \in \mathcal{S}$ such that $H \equiv_\Pi G[X]$. We show how to compute H from X. Let $k_{max} = (6c')^p$ where p is the smallest constant that satisfies the conditions for Π or $\overline{\Pi}$ being quasi-compact. For every $G_1 = (V_1, E_1) \in \mathcal{S}$, $G_2 = (V_2, E_2) \in \mathcal{S}$ and $k' \leq k_{max}$ we compute whether $(G_1 \oplus G_2, k') \in \Pi$. For each such triple the computation can be done in time $O((|V_1| + |V_2|)^p)$ since Π has finite integer index [25, 41]. Now, for every $G_1 \in \mathcal{S}$ and $k' \leq (|X| + |V_1|)^p$ we compute whether $(G[X] \oplus G_1, k') \in \Pi$. When all these computations are done, the results are stored in a table.

It is not hard to see that $H \equiv_\Pi G[X]$ if and only if there exists a constant c such that for all $G_2 \in \mathcal{S}$ and $k' \leq k_{max}$, $(H \oplus G_2, k') \in \Pi \Leftrightarrow (G[X] \oplus G_2, k' + c) \in \Pi$. Also, c is the constant such that for all r-boundaried graphs G_2 and and integers k', $(H \oplus G_2, k') \in \Pi \Leftrightarrow (G[X] \oplus G_2, k' + c) \in \Pi$. For each $H \in \mathcal{C}$ we can check whether $H \equiv_\Pi G[X]$ using this condition and the pre-computed table, and if $H \equiv_\Pi G[X]$, find the constant c.

After we have found a $H \in \mathcal{S}$ and the corresponding constant c, such that $H \equiv_\Pi G[X]$, we make G^* from G by replacing the extended r-protrusion X with H. Also, we set $k^* = k - c$. Since $|X| > c$ and H has at most c vertices, $|V^*| < |V|$. By the choice of H and c, $(G^*, k^*) \in \Pi$ if and only if $(G, k) \in \Pi$. This concludes the proof. ∎

8.1.2 Decomposition Theorems

Definition 8.1.12 *We say that a graph $G = (V, E)$ is (α, β, γ)-structured around S if $|S| \leq \alpha$ and V can be partitioned into $S, C_1, C_2, \ldots, C_\gamma$ such that $N_G(C_i) \subseteq S$, and $\max\{|N_G(C_i)|, \mathbf{tw}(G[C_i \cup N_G(C_i)])\} \leq \beta$, for every $i = 1, \ldots, \gamma$.*

Let $G = (V, E)$ be a graph embedded in some surface Σ. A *noose* in G is a closed curve N of Σ meeting only the vertices of G, we denote these vertices by $V_N = V \cap N$. We also define the *length* of N as the number of vertices it meets and we denote it by $|N|$, that is, $|N| = |V_N|$. The *face-width* of G is the minimum length of a non-contractible noose in G. We denote the Euler genus of a surface Σ by $\mathbf{eg}(\Sigma)$.

Lemma 8.1.13 *Let $G = (V, E)$ be a graph in $\mathcal{G}_g$ and let $S \subseteq V$ such that $\mathbf{B}_G^r(S) \leq q$. Then, $\mathbf{tw}(G) \leq 4(2r + 1)\sqrt{q + 2g} + 8r + 12g$.*

Proof. The result follows closely the argument of the proof of [44, Theorem 3.2] that, in turn, is based on [44, Lemma 3.1] about the distribution of every such set S in the interior of a $(\rho \times \rho)$-grid. As now we have graphs of higher genus we have to apply [47, theorem 4.7] and find a lower bound to the size of S in the graph obtained by a $(\rho \times \rho)$-grid after adding $O(g)$ edges. ∎

From now on, we set $f(r,g) = 4(2r+1)\sqrt{2+2g} + 8r + 12g$.

Lemma 8.1.14 *Let $G = (V, E)$ be a graph, embedded in a surface Σ of Euler genus g such that either $g = 0$ or the face-width of G is strictly greater than $4r+2$. Let $S \subseteq V$ be a set containing at least 3 vertices where $\mathbf{B}^r_G(S) = V$. Then there exists a set $S' \subseteq V$ such that $S \subseteq S'$ and G is $(r \cdot (4r+2) \cdot (3|S| - 6 + 6g), f(r,g), r \cdot (3|S| - 6 + 6g))$-structured around S'.*

Proof. We first need the following claim.
Claim: Let $G = (V, E)$ be a graph, embedded in a surface Σ of Euler genus g such that either $g = 0$ or the face-width of G is more than $4r + 2$. Let S be a set of at least 3 vertices such that $\mathbf{B}^r_G(S) = V$. Then there is a collection $\mathcal{R}$ of closed subsets of Σ, also known as *regions*, such that

- If two sets in $\mathcal{R}$ have common points, then these points lay on their boundaries.
- $\bigcup_{R \in \mathcal{R}} R \cap V = V$.
- The boundary of each $R \in \mathcal{R}$ is the union of two paths of length $\leq 2r+1$ between two vertices of S called the *anchors* of R. We denote the set of vertices on the boundary of R by $\mathbf{bor}(R)$.
- For each $R \in \mathcal{R}$ with u and v as anchors it holds that $R \cap V \subseteq \mathbf{B}^r_G(\{u, v\})$.
- For each $R \in \mathcal{R}$ with u and v as anchors it holds that $S \cap R = \{u, v\}$.
- $|\mathcal{R}| \leq r \cdot (3|S| + 6g - 6)$.

The above claim follows from [79, Lemma 1] for the case where $g = 0$. For the sake of completeness, we briefly present this proof together with its natural extension for embeddings of higher genus provided that the face-width of G is $> 4r + 2$. A collection $\mathcal{R}$ of closed subsets of Σ is constructed by a greedy algorithm as follows: *Start with an empty $\mathcal{R}$ and, as long as there are vertices not contained in some region R in $\mathcal{R}$, find a area-maximal region R defined by two paths of length $\leq 2r+1$ between two vertices in S (its anchors) that do not contain any vertex in S and add it to $\mathcal{R}$.* Notice that such a region R always exists even for non-planar embeddings, provided that the face-width of G is bigger than $4r+2$, as this region is always inside a big enough disk of the surface. To prove the claimed upper bound on the size of $\mathcal{R}$, consider a multigraph $G_{\mathcal{R}}$ (again embedded in Σ) with vertex set S and there is an edge between $u, v \in S$ whenever u and v are the anchors of some region $R \in \mathcal{R}$. Notice that $G_{\mathcal{R}}$ is also embedded in Σ and has face-width bigger than 1. This in turn implies that if two vertices u, v are joined by many copies of the same edge in $G_{\mathcal{R}}$, then a pair of these edges will define a disk Δ in Σ containing in its interior all other copies. Moreover, we may assume that $\Sigma \setminus \Delta$ contains a vertex w of S (this is necessary in case G has no planar embedding). We claim that the following *thinness property* holds: if there are $x \geq 2r$ copies of the edge $\{u, v\}$, then there exists a vertex $w' \in S$ laying in the interior of one of the the $x-1 \geq 2r-1$ area minimal disks $\Delta_1, \ldots, \Delta_x$ defined by the copies of $\{u, v\}$ inside Δ (the order is chosen such that consecutive disks have common edges). We prove this using the argument of the proof of [79, Lemma 1]. Assume to the contrary and consider the disk Δ_r. Indeed, by area-maximally of the choice of the regions in the above greedy procedure, it follows that Δ_r contains a vertex $z \in V$ whose distance in G from u and v and every other vertex in S is bigger than r, a contradiction. This implies that there exists a vertex $w' \in S$ laying in the interior of one of the the $x-1 \geq 2r-1$ area minimal disks $\Delta_1, \ldots, \Delta_x$ defined by the copies of $\{u, v\}$ inside Δ. Using the thinness property of $G_{\mathcal{R}}$ (see also [5, Lemma 5]) along with the Euler formula for graphs embedded in higher genus surfaces, we derive the claimed bound for $|\mathcal{R}|$ and the claim follows.

To complete the proof of the lemma, we define $S' = \bigcup_{R \in \mathcal{R}} \mathbf{bor}(R) \cap V$, that is, S' contains all the vertices belonging to the boundary of the sets in $\mathcal{R}$. As each such boundary is the union of two (u, v)-paths of length $\leq 2r+1$ where $u, v \in S$, we have that such a boundary can have at most $4r+2$ vertices. As $|\mathcal{R}| \leq r \cdot (3|S| + 6g - 6)$, we obtain that $|S'| \leq r \cdot (4r+2) \cdot (3|S| - 6 + 6g)$. For each $R_i \in \mathcal{R}$ assign a set $\mathcal{C}_i$ containing each connected component $C = (V_C, E_C)$ of $G \setminus S'$ for which $N_G(V_C) \subseteq \mathbf{bor}(R)$. If a connected component of $G \setminus S'$ can be assigned to more than one R_i, break ties arbitrarily so that $\{\mathcal{C}_i, i = 1, \ldots, |\mathcal{R}|\}$ forms a partition of the set of the connected components of $G \setminus S'$. Notice that for each $C \in \mathcal{C}_i$, $N_G(V_C) \subseteq R_i \subseteq S'$ and if we set $C_i = \cup_{C \in \mathcal{C}_i} V_C$ we have that $N_G(C_i) \subseteq R_i \subseteq S'$ and thus $|N_G(C_i)| \leq 4r + 2 \leq f(r, g)$ (for $i = 1, \ldots, |\mathcal{R}|$). It remains to prove that if $J_i = G[C_i \cup N_G(C_i)]$ then $\mathbf{tw}(J_i) \leq f(r, g)$. For this, recall that $R \cap V \subseteq \mathbf{B}^r_G(\{u, v\})$ and from Lemma 8.1.13, we obtain that $\mathbf{tw}(J_i) \leq f(r, g)$, (for $j = 1, \ldots, |\mathcal{R}|$). ∎

We are now in position to prove the following.

Lemma 8.1.15 *Let $G = (V, E)$ be a graph, embedded in a surface Σ of Euler genus g and let $S \subseteq V$, $|S| \geq 3$ such that $\mathbf{R}^r_G(S) = V$ for some $r \geq 0$. Then there exists a set S' such that $S \subseteq S'$ and G is $(h(r, g) \cdot |S|, h(r, g), h(r, g) \cdot |S|)$-structured around S', where $h(r, g) = O(rg)$.*

Proof. We use induction on g. In particular, we prove that there exists a set $S' \subseteq V$ such that G is $(\alpha_{g,|S|}, \beta_g, \gamma_{g,|S|})$-structured around S' where, for $g = 0$, we set $\alpha_{0,x} = r \cdot (4r+2) \cdot (3x - 6)$, $\beta_0 = f(r, 0)$ and $\gamma_{0,x} = r \cdot (3x - 6)$ and for $g \geq 1$

we have $\alpha_{g,x} = (2g-1)\cdot r\cdot 6(4r+2)^2 + r\cdot(4r+2)\cdot(3x+6g-6)$, $\beta_g = f(r,g)$ and $\gamma_{g,x} = (2g-1)\cdot r\cdot 6(4r+2) + r\cdot(3x+6g-6)$. Once we have proved this, the lemma follows by suitably choosing the function h. For our proof we distinguish two cases:

Case 1. $g = 0$ or the embedding of G has face-width $> 4r+2$. Then we draw G together with its radial graph $R_G = (V_R, E_R)$ and denote it by $\overline{G} = G \cup R_G = (\overline{V}, \overline{E})$, the superposition of the two drawings. By the definition of the radial graph, it follows that $\mathbf{B}^r_{\overline{G}}(S) = \overline{V}$. Clearly, as G is a subgraph of $\overline{G}$, the result will follow if we prove that $\overline{G}$ is $(\alpha_{g,|S|}, \beta_g, \gamma_{g,|S|})$-structured around some set $S' \subseteq V_R$. If G is embeddable in the sphere, then $\overline{G}$ is also embeddable in the sphere and if G is embeddable in a surface Σ of Euler genus $g > 4r+2$ then the face-width of $\overline{G}$ is also greater than $4r+2$. Therefore, in either case we have all the conditions required to apply Lemma 8.1.14 and hence the claim follows by applying Lemma 8.1.14.

Case 2. There is a non-contractible noose N in Σ of length at most $4r+2$. Then we split the graph along the vertices V_N of the noose and we distinguish two subcases depending if N is a surface separating or not.

Subcase 2.1. If N is a surface separating noose, then the splitting of the vertices of N creates two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ embedded in surfaces Σ_1 and Σ_2 respectively such that if $\mathbf{eg}(\Sigma_i) = g_i, i = 1,2$ then $g_1 + g_2 \leq g$ and $g_1 \cdot g_2 > 0$. Let S_i consists of the vertices in $S \cap V_i$ and all the (duplicated) vertices met by N. Notice that $\mathbf{R}^r_{G_i}(S_i) = V_i$ and that $|S_1| + |S_2| \leq 2|N| + |S| \leq 2(4r+2) + |S|$.

By the induction hypothesis, G_i is $(\alpha_{g_i,|S_i|}, \beta_{g_i}, \gamma_{g_i,|S_i|})$-structured around some set S'_i where $S_i \subseteq S'_i$ and $g_i > 1, i = 1,2$. Let G^+ be the disjoint union of G_1 and G_2 and observe that G^+ is $(\alpha_{g_1,|S_1|} + \alpha_{g_2,|S_2|}, \max\{\beta_{g_1}, \beta_{g_2}\}, \gamma_{g_1,|S_1|} + \gamma_{g_2,|S_2|})$-structured around $S'_1 \cup S'_2$. Notice that

$$\begin{aligned}
\alpha_{g_1,|S_1|} + \alpha_{g_2,|S_2|} &\leq (2g_1 - 1 + 2g_2 - 1)\cdot r\cdot 6(4r+2)^2 + \\
&\quad r\cdot(4r+2)(3|S_1| + 3|S_2| + 6g_1 + 6g_2 - 12) \\
&\leq (2g_1 + 2g_2 - 2)\cdot r\cdot 6(4r+2)^2 + \\
&\quad r\cdot(4r+2)(3(2(4r+2) + |S|) + 6g - 6) \\
&\leq (2g_1 + 2g_2 - 1)\cdot r\cdot 6(4r+2)^2 - r\cdot 6(4r+2)^2 + \\
&\quad r\cdot 6(4r+2)^2 + r\cdot(4r+2)\cdot(3|S| + 6g - 6) \\
&\leq (2g-1)\cdot r\cdot 6(4r+2)^2 + r\cdot(4r+2)\cdot(3|S| + 6g - 6) \\
&= \alpha_{g,|S|}
\end{aligned}$$

Similarly, we can show that $\gamma_{g_1,|S_1|} + \gamma_{g_2,|S_2|} \leq \gamma_{g,|S|}$ and, as $\max\{\beta_{g_1}, \beta_{g_2}\} \leq \beta_g$, we conclude that G^+ is $(\alpha_{g,|S|}, \beta_g, \gamma_{g,|S|})$-structured around $S'_1 \cup S'_2$. As all the duplicated vertices of V_N are in $S_1 \cup S_2$, we can identify back these duplicated vertices occuring in S_1 and S_2 and obtain that G is $(\alpha_{g,|S|}, \beta_g, \gamma_{g,|S|})$-structured around some set S'.

Subcase 2.2. If N is not a surface separating noose, then the splitting of the vertices of N creates a new graph $G_0 = (V_0, E_0)$ embedded in a surface Σ_0 of Euler genus g_0 where $g_0 < g$. Notice that if S_0 is the set of vertices in G_0 consisting of the non-duplicated vertices of S and all the duplicated vertices, then $\mathbf{R}^r_{G_0}(S_0) = V_0$ and $|S_0| \leq 2|N| + |S| \leq 2(4r+2) + |S|$. From the induction hypothesis G_0 is $(\alpha_{g_0,|S_0|}, \beta_{g_0}, \gamma_{g_0,|S_0|})$-structured around some set S'_0 where $S_0 \subseteq S'_0$. Notice that

$$\begin{aligned}
\alpha_{g_0,|S_0|} &\leq (2g_0 - 1)\cdot r \cdot 6(4r+2)^2 + r\cdot(4r+2)\cdot(3|S_0| + 6g_0 - 6)\\
&\leq (2(g-1)-1)\cdot r\cdot 6(4r+2)^2 + r\cdot(4r+2)\cdot(3(2(4r+2)+|S|) + 6g - 6)\\
&\leq (2g-1)\cdot r\cdot 6(4r+2)^2 - 2\cdot r\cdot 6(4r+2)^2 +\\
&\quad r\cdot 6(4r+2)^2 + r\cdot(4r+2)\cdot(3|S| + 6g - 6)\\
&\quad (2g-1)\cdot r\cdot 6(4r+2)^2 + r\cdot(4r+2)\cdot(3|S| + 6g - 6)\\
&= \alpha_{g,|S|}
\end{aligned}$$

and similarly $\gamma_{g_0,|S_0|} \leq \gamma_{g,|S|}$. As $\beta_{g_0} \leq \beta_g$, we conclude that G_0 is $(\alpha_{g,|S|}, \beta_g, \gamma_{g,|S|})$-structured around S'_0. As all the duplicated vertices of V_N are in S_0, we can identify them back and deduce that G is $(\alpha_{g,|S|}, \beta_g, \gamma_{g,|S|})$-structured around some set S'. This concludes the proof. ■

8.1.3 Kernels

Armed with the tools developed in the previous two sections, we are now ready to prove Theorems 8.1.1 and 8.1.3. We say that an instance (G', k') of a parameterized problem Π is *reduced with respect to a set $\mathcal{Q}$ of reduction rules* if none of the reduction rules in $\mathcal{Q}$ can be applied to (G', k').

Proof of Theorem 8.1.1

Proof. We first give a proof for the case when Π is compact and Π^α is an annotated p-MIN-CMSO problem. A proof for the case when Π is compact and Π^α is an annotated p-EQ/MAX-CMSO problem is identical.

We know that Π is compact and hence there exists an integer r such that for all $(G = (V, E), k) \in \Pi^\alpha$, there is an embedding of G into a surface of genus at most g, and a set $S \subseteq V$ such that $\mathbf{R}^r_G(S) = V$ and $|S| \leq r \cdot k$. We show that for all $(G, k) \in \Pi^\alpha$, the equivalent instance $(G' = (V', E'), k)$, reduced with respect to the reduction rule given by Lemma 8.1.8, has $|V'| = O(k^2)$. Since $(G' = (V', E'), k) \in \Pi^\alpha$ and Π is compact, there exists an embedding of G' into a surface of genus at most g and a set $S \subseteq V'$ such that $\mathbf{R}^r_{G'}(S) = V'$. Hence by applying Lemma 8.1.15 we obtain a set S' such that G' is (α, β, γ)-structured around S', where $\alpha, \gamma = O(rg|S|)$ and $\beta = O(rg)$. This implies that V' can be partitioned into $S', C_1, C_2, \ldots, C_\gamma$ such that $N_{G'}(C_i) \subseteq S'$, $|N_{G'}(C_i)| \leq \beta$ and $\mathbf{tw}(G'[C_i \cup N_G(C_i)]) \leq \beta$ for every $i \leq \gamma$. Observe that each C_i is a β-protrusion in G' and $|C_i \cup N_G(C_i)| \leq ck$, where c is a constant of Lemma 8.1.8, otherwise we could have applied Lemma 8.1.8. This implies that

$$|V'| \leq |S'| + \sum_{i=1}^{\gamma} |C_i| = O\left(rg|S| + \sum_{i=1}^{\gamma} ck\right) = O(k^2),$$

for some fixed g and r. Here constants hidden in big-Oh depend only on r and g.

So given an input (G, k), if the size of the reduced graph is more than c^*k^2 for some constant c^* then we return NO else we have G' as the desired annotated kernel for G.

Now we give a proof for the case when $\overline{\Pi}$ is compact and Π^α is an annotated p-MAX-CMSO problem. A proof for the case when $\overline{\Pi}$ is compact and Π^α is an annotated p-MIN/EQ-CMSO problem is similar. Towards this end, we claim that for all $(G, k) \in \overline{\Pi}^\alpha$, the equivalent instance $(G' = (V', E'), k)$, reduced with respect to the reduction rule given by Lemma 8.1.10, has $|V'| = O(k^2)$. The proof for the claim is identical to the one we gave above to bound all the YES instance for an annotated compact p-MIN-CMSO problem. So given an input (G, k), if the equivalent instance $(G' = (V', E'), k)$, reduced with respect to the reduction rule given by Lemma 8.1.10, is more than c^*k^2 for some constant c^* then we return YES else we have G' as the desired annotated kernel for G. The reason we return YES is that if (G, k) would have a NO instance then the size of $|V'| \leq c^*k^2$ as $(G, k) \in \overline{\Pi}^\alpha$. ∎

Proof of Corollary 8.1.2

Proof. We know that Π is NP-complete and the annotated version Π^α is in NP. So given an instance $(G = (V, E), k)$, we apply Theorem 8.1.1 on the annotated instance $(G = (V, E), V, k)$, that is we take V as Y, the set of black vertices. If we get YES or NO as an answer then we return the same. Else for $(G' = (V', E'), k)$ of size polynomial in k, we apply polynomial time many to one reduction from Π^α to Π on G' and obtain a graph $G'' = (V'', E'') \in \mathcal{G}_g$ and an integer k' such that $|V''|, k' \leq k^{O(1)}$ and $(G', k) \in \Pi^\alpha$ if and only if $(G'', k') \in \Pi$. This implies that in this case we have polynomial kernel for Π. ∎

Proof of Theorem 8.1.3

The idea of the proof is that if a problem has finite integer index then an instance reduced with respect to the reduction rule given by Lemma 8.1.11 has bounded radial distance. We first prove a lemma which will assist us in proving Theorem 8.1.3.

Lemma 8.1.16 *Let $G = (V, E)$ be a graph embedded into a surface of genus at most g and $S \subseteq V$ such that $\mathbf{tw}(G \setminus \mathbf{R}^r_G(S)) \leq r$ and all r-protrusions of G have size at most m. Then there exists a $S' \subseteq V$ such that $|S'| \leq |S| + g \cdot (m + 2r + 1)$ and $V = \mathbf{R}^{m+3r+1}_G(S')$.*

Proof. We prove the lemma using induction on Euler genus g of the graph $G \setminus \mathbf{R}^r_G(S)$. We distinguish two cases:

Case 1. $g = 0$ or the embedding of $G \setminus \mathbf{R}^r_G(S)$ has face-width $> m + 2r + 1$. We claim that $V = \mathbf{R}^{m+3r+1}_G(S)$, that is, $S' = S$. For this assume, towards a contradiction, that $x \in V \setminus \mathbf{R}^{m+3r+1}_G(S)$ and consider the subgraph J of G induced by the set $\mathbf{R}^{m+2r+1}_G(x)$, that is, the vertices of G that are within radial distance at most $m + 2r + 1$ from x. Notice that J is a subgraph of $G \setminus \mathbf{R}^r_G(S)$, therefore $\mathbf{tw}(J) \leq r$. As either $g = 0$ or because the face-width of G is $> m + 2r + 1$, all the vertices and edges of J are embedded inside a closed disk in Σ. Moreover, there exist $m + 2r + 1$ nested disjoint cycles $C_1, \ldots, C_{m+2r+1}$ with vertex sets $V_1, \ldots, V_{m+2r+1}$ respectively in R_G such that, if Δ_i is the closed disk with C_i as its border and contains x then $i < j$ implies that $\Delta_i \subset \Delta_j$. For $i = 1, \ldots, m+2r+1$, V_i contains vertices and faces whose radial distance from x, in G, is exactly i. We need the following claim.

Claim. Each cycle of the radial graph R_G that is entirely in $(\Delta_{m+r+1} \setminus \Delta_m)$ and separates S and x, has length $> 2r$.

Proof. Indeed, if this is not the case for some cycle C with vertex set V_C, then $L = V \cap V_C$ is a separator of G where $|L| \leq r$ and such that the connected component, say F, of $G \setminus L$ that contains x is a subgraph of J. Then $\mathbf{tw}(F) \leq \mathbf{tw}(J) \leq r$ and F is an r-protrusion of G. As F contains x and has more than m vertices, it is a contradiction to the assumption that all r-protrusions of G have size at most m. ■

Applying the above claim to the cycles $C_{m+1}, \ldots, C_{m+2r+1}$ we obtain that they all have length $> 2r$. We now construct an auxiliary graph R^* by taking $R_G \cap (\Delta_{m+r+1} \setminus \Delta_m)$, adding a vertex s adjacent to all vertices of C_{m+1} and adding a vertex t adjacent to all vertices in C_{m+2r+1}. We claim that there is no (s, t)-separator in R^* of size $\leq 2r$. Indeed, such a separator would imply the existence of a cycle C in R_G of size $\leq 2r$, a contradiction to the above claim. By Menger's theorem, it follows that there are $> 2r$ disjoint (s, t)-paths in R^* that correspond to $> 2r$ disjoint paths from the vertices of C_{m+1} to the vertices of C_{m+2r+1}. The intersection of these paths with cycles $C_{m+1}, \ldots, C_{m+2r+1}$, imply the existence of a $(2r + 1) \times (2r + 1)$-grid as a minor of R_J, where R_J is the radial graph corresponding to J. Therefore, $\mathbf{tw}(R_J) > 2r$ (from [19, Lemma 88]), which, using [98, Lemma 3], implies that $\mathbf{tw}(J) > r$, a contradiction.

Case 2. There is a non-contractible noose N, meeting the vertices of V in V_N, in $G \setminus \mathbf{R}^r_G(S)$ of length at most $r' = m + 2r + 1$ (assume that $G \setminus \mathbf{R}^r_G(S)$ is embedded in a surface Σ). Let $S'' = S \cup V_N$. Observe that $\mathbf{tw}(G \setminus \mathbf{R}^r_G(S'')) \leq r$

and all r-protrusions of G have size at most m. Furthermore let Σ' be the surface in which $G \setminus \mathbf{R}^r_G(S'')$ can be embedded. Then $\mathbf{eg}(\Sigma') < \mathbf{eg}(\Sigma) \leq g$. Hence by induction hypothesis, there exists a set S' such that

$$|S'| \leq |S''|+(g-1)(m+2r+1) \leq |S|+|V_N|+(g-1)(m+2r+1) \leq |S|+g(m+2r+1),$$

and $V = \mathbf{R}^{m+3r+1}_G(S')$. This concludes the proof. ■

We are now in position to give the proof of Theorem 8.1.3.

Proof. Let us assume that Π is quasi-compact. Fix $t = c^*rg$ where c^* is a constant to be defined later. Let $(G' = (V', E'), k')$, $k' \leq k$, be a reduced instance with respect to reduction rule given by Lemma 8.1.11. Hence there is no extended t-protrusion of size more than c, where c is a constant appearing in the statement of Lemma 8.1.11 and $(G, k) \in \Pi$ if and only if $(G', k') \in \Pi$ and $G' \in \mathcal{G}_g$. Hence, what remains to show is that $|V'| = O(k)$. The proof for this is similar to the one given for Theorem 8.1.1.

Now we show that if $(G' = (V', E'), k') \in \Pi$ then $|V'| \leq O(k)$. Since Π is quasi-compact and $(G', k') \in \Pi$, there is an embedding of G into a surface of genus at most g and a set $S \subseteq V'$ such that $|S| \leq r \cdot k'$ and $\mathbf{tw}(G' \setminus \mathbf{R}^r_{G'}(S)) \leq r$. Since all t-protrusions of G' are of size most c we have that all r-protrusions of G' are of size at most c. Hence by Lemma 8.1.16 there exists a set S' such that $|S'| \leq |S| + g \cdot (c+2r+1)$ and $V' = \mathbf{R}^{3r+c+1}_{G'}(S')$. Given S', we apply Lemma 8.1.15 and obtain a set S'' such that G' is (α, β, γ)-structured around S'', where $\alpha, \gamma = O(rg|S'|)$ and $\beta = O(rg)$. This implies that V' can be partitioned into $S'', C_1, C_2, \ldots, C_\gamma$ such that $N_{G'}(C_i) \subseteq S''$, $|N_{G'}(C_i)| \leq \beta$ and $\mathbf{tw}(G'[C_i \cup N_{G'}(C_i)]) \leq \beta$ for every $i \leq \gamma$. Fix c^* such that $t \geq \beta$. This implies that $G'[C_i \cup N_{G'}(C_i)]$ is a t-protrusion of G' and hence its size is bounded by c. Now we are ready to bound the size of V'.

$$|V'| \leq |S''| + \sum_{i=1}^{\gamma} |C_i| \leq |S''| + \gamma \cdot \beta \cdot c = O(r^3 g^3 c^2 k) = O(k),$$

for fixed r, g and c. So given an input (G, k), if the size of the reduced graph is more than $\tilde{c} \cdot k$ for some $\tilde{c}$, we return NO else we have G' as the desired kernel.

The proof for the case when $\overline{\Pi}$ is quasi-compact is similar to the proof we gave for the case when $\overline{\Pi}$ was compact and Π^α was an annotated p-MAX-CMSO problem in Theorem 8.1.1. This concludes the proof. ■

8.1.4 Implications of Our Results

In this section we mention a few parameterized problems for which we can obtain either polynomial or linear kernel using either Theorem 8.1.1 or Theorem 8.1.3. Various other problems for which we can obtain either polynomial or linear kernels using our results are mentioned in appendix.

A Sufficient Condition for Finite Integer Index

We first give a sufficient condition which implies that a large class of p-MIN/MAX-CMSO problems has finite integer index. We prove it here for vertex versions of p-MIN/MAX-CMSO problems that is if Π is a p-MIN/MAX-CMSO problem then P_Π is a property of vertex sets. The edge version can be dealt in a similar manner.

Let Π be a p-MIN-CMSO problem and $\mathcal{F}_t$ be the set of pairs $(G = (V, E), S)$ where G is a t-boundaried graph and $S \subseteq V$. For a t-boundaried graph $G = (V, E)$ we define the function $\zeta_G : \mathcal{F}_t \to \mathbb{N} \cup \{\infty\}$ as follows. For a pair $(G' = (V', E'), S') \in \mathcal{F}_t$, if there is no set $S \subseteq V$ $(S \subseteq E)$ such that $P_\Pi(G \oplus G', S \cup S')$ holds, then $\zeta_G((G', S')) = \infty$. Otherwise $\zeta_G((G', S'))$ is the size of the smallest $S \subseteq V$ $(S \subseteq E)$ such that $P_\Pi(G \oplus G', S \cup S')$ holds. If Π is a p-MAX-CMSO problem then we define $\zeta_G((G', S'))$ to be the size of the largest $S \subseteq V$ $(S \subseteq E)$ such that $P_\Pi(G \oplus G', S \cup S')$ holds. If there is no set $S \subseteq V$ $(S \subseteq E)$ such that $P_\Pi(G \oplus G', S \cup S')$ holds then $\zeta_G((G', S')) = \infty$.

Definition 8.1.17 *A p-MIN-CMSO problem Π is said to be* strongly monotone *if there exists a function $f : \mathbb{N} \to \mathbb{N}$ such that the following condition is satisfied. For every t-boundaried graph $G = (V, E)$, there is a subset $S \subseteq V$ such that for every $(G' = (V', E'), S') \in \mathcal{F}_t$ such that $\zeta_G((G', S'))$ is finite, $P_\Pi(G \oplus G', S \cup S')$ holds and $|S| \leq \zeta_G((G', S')) + f(t)$.*

Definition 8.1.18 *A p-MAX-CMSO problem Π is said to be* strongly monotone *if there exists a function $f : \mathbb{N} \to \mathbb{N}$ such that the following condition is satisfied. For every t-boundaried graph $G = (V, E)$, there is a subset $S \subseteq V$ such that for every $(G' = (V', E'), S') \in \mathcal{F}_t$ such that $\zeta_G((G', S'))$ is finite, $P_\Pi(G \oplus G', S \cup S')$ holds and $|S| \geq \zeta_G((G', S')) - f(t)$.*

Lemma 8.1.19 *Every strongly monotone p-MIN-CMSO and p-MAX-CMSO problem has finite integer index.*

Proof. We prove for p-MIN-CMSO problems, the proof for p-MAX-CMSO is similar. Let Π be a monotone p-MIN-CMSO problem. Then P_Π is a finite state property of t-boundaried graphs with a distinguished vertex set S [27, 39]. In particular for every t, there exists a finite set $\mathcal{S}$ of pairs (G, S) such that $G = (V, E)$ is a t-boundaried graph and $S \subseteq V$ such that the set $\mathcal{S}$ satisfies the following properties. For any t-boundaried graph $G_1 = (V_1, E_1)$ and set $S_1 \subseteq V_1$ there is a pair $(G_2 = (V_2, E_2), S_2) \in \mathcal{S}$ such that for every t-boundaried graph $G_3 = (V_3, E_3)$ and set $S_3 \subseteq V_3$ we have that $P_\Pi(G_1 \oplus G_3, S_1 \cup S_3) \iff P_\Pi(G_2 \oplus G_3, S_2 \cup S_3)$. We fix such a set $\mathcal{S}$.

For a t-boundaried graph $G = (V, E)$ we define the *signature* $\zeta_G^{\mathcal{S}} : \mathcal{S} \to \mathbb{N} \cup \{\infty\}$ of G to be ζ_G with domain restricted to $\mathcal{S}$. We now argue that for any t-boundaried graph G, the maximum finite value of $\zeta_G^{\mathcal{S}}$ is at most $f(t)$ larger than the minimum finite value taken by $\zeta_G^{\mathcal{S}}$. Since Π is strongly monotone there exists a subset S of V that satisfies the conditions of Definition 8.1.17. Let $(G' = (V', E'), S') \in \mathcal{S}$ such that $\zeta_G^{\mathcal{S}}((G', S'))$ is finite. Then $P_\Pi(G \oplus G', S \cup S')$ holds and hence $\zeta_G^{\mathcal{S}}((G', S')) \leq |S|$. Furthermore the conditions of Definition 8.1.17 imply that $|S| - f(t) \leq \zeta_G^{\mathcal{S}}((G', S')) \leq |S|$. Hence the minimum and the maximum finite values of $\zeta_G^{\mathcal{S}}$ can differ by at most $f(t)$. By the pigeon hole principle there is a finite set $\mathcal{R}$ of t-boundaried graphs such that for any t-boundaried graph G there is a $G_R \in \mathcal{R}$ and a constant c_R depending only on the size of G and G_R, such that for all $(G', S') \in \mathcal{S}$ we have $\zeta_{G_R}((G', S')) + c_R = \zeta_G((G', S'))$.

We now argue that $\mathcal{R}$ forms a set of representatives for (Π, t). Let $G = (V, E)$ be a t-boundaried graph and let $G_R = (V_R, E_R) \in \mathcal{R}$ and c_R be a constant such that for all $(G', S') \in \mathcal{S}$ we have $\zeta_{G_R}((G', S')) + c_R = \zeta_G((G', S'))$. Let $G' = (V', E')$ be a t-boundaried graph and k be an integer such that $(G \oplus G', k) \in \Pi$. We argue that $(G_R \oplus G', k - c_R) \in \Pi$. Let $Z \subseteq V \cup V'$ be a minimum size set such that $P_\Pi(G \oplus G', Z)$ is satisfied, $Z' = Z \cap V'$ and $Z_G = Z \setminus Z'$. Since $(G \oplus G', k) \in \Pi$ we have that $|Z| \leq k$ and hence $\zeta_G(G', Z \cap V')$ is finite. Let $(G_{\mathcal{S}}, Z_{\mathcal{S}}) \in \mathcal{S}$ be the representative of (G', Z'). Then $P_\Pi(G \oplus G_{\mathcal{S}}, Z_G \cup Z_{\mathcal{S}})$ holds and $|Z_G| = \zeta_G(G_{\mathcal{S}}, Z_{\mathcal{S}})$. Now we have that $\zeta_{G_R}((G_{\mathcal{S}}, Z_{\mathcal{S}})) + c_R = \zeta_G((G_{\mathcal{S}}, Z_{\mathcal{S}}))$. Hence, there is a set $S_R \subseteq V_R$ of size $|Z_G| - c_R$ such that $P_\Pi(G_R \oplus G_{\mathcal{S}}, S_R \cup Z_{\mathcal{S}})$ holds. Then, since $(G_{\mathcal{S}}, Z_{\mathcal{S}})$ is the representative of (G', Z') we have that $P_\Pi(G_R \oplus G', S_R \cup Z')$ holds. Now we have that $|S_R \cup Z'| \leq |S_R| + |Z'| = |Z_G| - c_R + |Z'| = |Z| - c_R \leq k - c_R$. This implies that $(G_R \oplus G', k - c_R) \in \Pi$. The proof for the other direction that if $(G_R \oplus G', k - c_R) \in \Pi$ then $(G \oplus G', k) \in \Pi$ is symmetric. This concludes the lemma. ∎

Covering and Packing Problems

Minor Covering and Packing: We give below a few generic problems which subsumes many problems in itself and fit in our kernelization framework. Let $\mathcal{H}$ be a finite set of connected planar graphs.

Vertex-$\mathcal{H}$-Covering
Input: A graph $G = (V, E) \in \mathcal{G}_g$ and a non-negative integer k.
Question: Is there an $S \subseteq V$ that $|S| \leq k$ and $G[V \setminus S]$ does not contain any of the graphs in $\mathcal{H}$ as a minor?

Vertex-$\mathcal{H}$-Packing
Input: A graph $G \in \mathcal{G}_g$ and a non-negative integer k.
Question: Does there exist k vertex disjoint subgraphs $G_1, \ldots, G_k$ of G such that each of them contains some graph in $\mathcal{H}$ as a minor?

Lemma 8.1.20 *Let $\mathcal{H}$ be a finite set of connected planar graphs and let Π_1 denote* VERTEX-$\mathcal{H}$-PACKING. *Then* VERTEX-$\mathcal{H}$-COVERING *has finite integer index and is quasi-compact and* VERTEX-$\mathcal{H}$-PACKING *has finite integer index and $\overline{\Pi}_1$ is quasi-compact.*

Proof. Let Π_v denote VERTEX-$\mathcal{H}$-COVERING. We first show that Π_v is quasi compact. If $(G = (V, E), k) \in \Pi_v$ then we know that there exists a set $S \subseteq V$ such that $G[V \setminus S]$ does not contain any graph in $\mathcal{H}$ as a minor. Now we show that the treewidth of $G[V \setminus S]$ is at most a constant. To show this we need following results.

- Every planar graph $H = (V_H, E_H)$ is a minor of the $r \times r$ grid, where $r = 14|V_H| - 24$ [123].
- For any fixed graph H, every H-minor free graph that does not contain a $w \times w$ grid as a minor has treewidth $O(w)$ [45].

Let $q = \max\{|H| \mid H \in \mathcal{H}\}$ and $w = 14q - 24$. Then observe that $G[V \setminus S]$ does not contain $w \times w$ grid as a minor and hence $\mathbf{tw}(G[V \setminus S]) \leq O(w)$. This implies that Π_v is quasi-compact.

Next we show that $\overline{\Pi}_1$ is quasi-compact. We first introduce some definitions. Given a graph $G = (V, E)$, we define the *covering number of G with respect to the class $\mathcal{H}$*, $\mathbf{cov}_{\mathcal{H}}(G)$, as the minimum k such that there exists $S \subseteq V$ of size k such that $G[V \setminus S]$ does not contain any of the graphs in $\mathcal{H}$ as a minor. The *packing number of G with respect to the class $\mathcal{H}$*, is defined as,

$$\begin{aligned}\mathbf{pack}_{\mathcal{H}}(G) \quad = \quad & \max\{k \mid \exists \text{ a partition } V_1, \ldots, V_k \text{ of } V \text{ such that} \\ & \qquad \forall_{i\in\{1,\ldots,k\}} G[V_i] \text{ contains a graph in } \mathcal{H} \text{ as a minor}\}.\end{aligned}$$

Less formally, $\mathbf{pack}_{\mathcal{H}}(G) \geq k$ if G contains k vertex-disjoint minors in $\mathcal{H}$. We need the following Erdős-Pósa type of result shown in [67] for our purpose.

Claim 8.1.21 *Let $\mathcal{H}$ be a finite set of connected planar graphs, $q = \max\{|H| \mid H \in \mathcal{H}\}$ and $\mathcal{G}$ be a non-trivial minor-closed graph class. Then there is a constant $\sigma_{\mathcal{G},q}$ depending only on $\mathcal{G}$ and q such that for every graph $G \in \mathcal{G}$, it holds that*

$$\mathbf{pack}_{\mathcal{H}}(G) \leq \mathbf{cover}_{\mathcal{H}}(G) \leq \sigma_{\mathcal{G},q} \cdot \mathbf{pack}_{\mathcal{H}}(G).$$

Using Claim 8.1.21 we show that $\overline{\Pi}_1$ is quasi-compact. Observe that if $(G, k) \in \Pi_1$ and $(G, k + 1) \notin \Pi_1$, then by Claim 8.1.21, $\mathbf{cover}_{\mathcal{H}}(G) = O(k)$. Hence by an argument, similar to the one used for showing that Π_v is quasi-compact, we have that $\overline{\Pi}_1$ is quasi-compact.

It is known that Π_v is a p-MIN-CMSO problem and Π_1 is a p-MAX-CMSO problem. Now using Lemma 8.1.19 we show that Π_v is finite integer index. We show that Π_v is strongly monotone. Given a t-boundaried graph $G = (V, E)$, with $\partial(G)$ as its boundary, let $S'' \subseteq V$ be a minimum set of vertices in G such that $G[V \setminus S'']$ does not contain any graph in $\mathcal{H}$ as a minor. Take $S = S'' \cup \partial(G)$. Now for any $(G' = (V', E'), S') \in \mathcal{F}_t$ such that $\zeta_G((G', S'))$ is finite we have that $G \oplus G'[(V \cup V') \setminus (S \cup S')]$ does not contain any graph in $\mathcal{H}$ as a minor and $|S| \leq \zeta_G((G', S')) + t$. This proves that Π_v is strongly monotone.

Next we show that Π_1 is finite integer index. Given a t-boundaried graph $G = (V, E)$, we define its signature, ζ_G, as follows. Let $\mathcal{F}_t^{qt} \subseteq \mathcal{G}_g$ be the set of all t-boundaried graph of size at most qt and $\zeta_G : \mathcal{F}_t^{qt} \to \mathbb{N}$. For every $G' \in \mathcal{F}_t^{qt}$ we define $\zeta_G(G') = \mathbf{pack}_{\mathcal{H}}(G \oplus G')$. For any $G', G'' \in \mathcal{F}_t^{qt}$, $|\zeta_G(G') - \zeta_G(G'')| \leq t + qt$, as $\zeta_G(\emptyset) \leq \zeta_G(G') \leq \zeta_G(\emptyset) + qt + t$ for all $G' \in \mathcal{F}_t^{qt}$. Hence, by the pigeon hole principle there is a finite set $\mathcal{R}$ of t-boundaried graphs such that for any t-boundaried graph G there is a $G_R \in \mathcal{R}$ and a constant c_R depending only on the size of G and G_R, such that $\zeta_{G_R} + c_R = \zeta_G$.

We now argue that $\mathcal{R}$ forms a set of representatives for (Π_1, t). Let $G = (V, E)$ be a t-boundaried graph and let $G_R = (V_R, E_R) \in \mathcal{R}$ and c_R be a constant such that $\zeta_{G_R} + c_R = \zeta_G$. Let $G' = (V', E')$ be a t-boundaried graph and k be an integer such that $(G \oplus G', k) \in \Pi_1$. We argue that $(G_R \oplus G', k - c_R) \in \Pi_1$. Let $\mathcal{S}$ be a set of k vertex disjoint minors in $G \oplus G'$ and H be a minor in $\mathcal{S}$ which goes across or touch the boundary $\partial(G)$. Now contract the part of this minor belonging to the side of G' as much close to the boundary $\partial(G)$ as possible. Since the size of largest graph in $\mathcal{H}$ is at most q, we have that the part of the H belonging to the side of G' can be contracted to the border except for some q vertices, which could possibly be hanging out of the boundary. We do this for every minor in $\mathcal{S}$ which is going across. Let $\mathcal{S}'$ be the set of minors resulting after the contraction operation has been performed. Now we take the boundary $\partial(G)$ and all the minors of $\mathcal{S}'$ hanging out of it, and call this resulting graph $\tilde{G} = (\tilde{V}, \tilde{E})$. Observe that we can have at most t minors in $\mathcal{S}'$ which can hang out of the border as these are vertex disjoint minors. Hence $|\tilde{V}| \leq qt$ and $\tilde{G}$ is a t-boundaried graph. Now we know that $\zeta_{G_R} + c_R = \zeta_G$ and hence $\zeta_{G_R}(\tilde{G}) + c_R = \zeta_G(\tilde{G})$. By definition $\zeta_{G_R}(\tilde{G}) = \mathbf{pack}_{\mathcal{H}}(G \oplus \tilde{G}) = \mathbf{pack}_{\mathcal{H}}(G \oplus G')$, and hence $(G_R \oplus G', k - c_R) \in \Pi_1$. The proof for the other direction that if $(G_R \oplus G', k - c_R) \in \Pi_1$, then $(G \oplus G', k) \in \Pi_1$ is symmetric. This concludes that Π_1 is finite integer index. ∎

Vertex-$\mathcal{H}$-Covering contains various problems as a special case, for example: (a) Feedback Vertex Set by taking $\mathcal{H} = \{\triangle\}$ where $\triangle$ is a cycle of length at most 3; deleting at most k vertices to obtain a graph of fixed treewidth t; deleting at most k vertices to obtain a graph into a graph class $\mathcal{M}$, where $\mathcal{M}$ is characterized by a finite set of minors. Similarly Vertex-$\mathcal{H}$-Packing contains problem like Cycle Packing as a special case.

Subgraph Covering and Packing:

Let $\mathcal{S}$ be a finite set of connected graphs.

VERTEX-$\mathcal{S}$-COVERING
Input: A graph $G \in \mathcal{G}_g$ and a non-negative integer k.
Question: Is there a $S \subseteq V$ such that $|S| \leq k$ and $G[V \setminus S]$ does not contain any of the graphs in $\mathcal{S}$ as a subgraph?

We similarly define EDGE-$\mathcal{S}$-COVERING by demanding $S \subseteq E$ in the above definition.

VERTEX-$\mathcal{S}$-PACKING
Input: A graph $G \in \mathcal{G}_g$ and a non-negative integer k.
Question: Does there exists k vertex disjoint subgraphs $G_1, G_2, ..., G_k$ in G such that, for all i G_i is isomorphic to a graph in $\mathcal{S}$.

We define EDGE-$\mathcal{S}$-PACKING by demanding that $G_1, \ldots, G_k$ be *edge disjoint* subgraphs of G.

We can not show that VERTEX/EDGE-$\mathcal{S}$-COVERING or VERTEX/EDGE-$\mathcal{S}$-PACKING or there no instances are compact unless we do the following simple preprocessing.

Redundant Vertex and Edge Rule: Given an input $(G = (V, E), k)$ to VERTEX/EDGE-$\mathcal{S}$-COVERING or VERTEX/EDGE-$\mathcal{S}$-PACKING remove all edges and vertices that are not part of any subgraph isomorphic to any graph in $\mathcal{S}$.

We can perform the Redundant Vertex and Edge Rule in $O(|V| \cdot |\mathcal{S}|)$ time by looking at a small ball around an edge e or a vertex v and check whether the ball contains a subgraph isomorphic to a graph in $\mathcal{S}$ and contains the edge e or the vertex v. This algorithm to check a subgraph isomorphic to a given graph containing a particular vertex or edge appears in a paper by Eppstein [55].

Lemma 8.1.22 *Let $\mathcal{S}$ be a finite set of connected graphs and Π_1 and Π_2 correspond to* VERTEX-$\mathcal{S}$-PACKING *and* EDGE-$\mathcal{S}$-PACKING *respectively. Then the following hold: (a)* VERTEX-$\mathcal{S}$-COVERING *has finite integer index and is compact; (b)* VERTEX-$\mathcal{S}$-PACKING *has finite integer index and $\overline{\Pi}_1$ is compact; (c)* EDGE-$\mathcal{S}$-COVERING *is p-*MIN*-CMSO problem and is compact; and (d)* EDGE-$\mathcal{S}$-PACKING *is p-*MAX*-CMSO problem and $\overline{\Pi}_2$ is compact.*

Proof. Let Π_v and Π_e denote VERTEX-$\mathcal{S}$-COVERING and EDGE-$\mathcal{S}$-COVERING respectively. Let $s = \max\{|V^*| \mid G^* = (V^*, E^*) \in \mathcal{S}\}$. Without loss of generality we assume that an input (G, k) to all these problems are *reduced* with respect to Redundant Vertex and Edge Rule.

We first show that these problems or their no instances are compact. Let $(G, k) \in \Pi_v$ then we know that there is a $S \subseteq V$ such that $|S| \leq k$ and $G[V \setminus S]$ does not contain any of the graphs in $\mathcal{S}$ as a subgraph and every vertex and edge is in some subgraph in G which is isomorphic to a subgraph in $\mathcal{S}$. This implies that every vertex in $u \in V \setminus S$ is in at most $r = O(s)$ distance away from a vertex in S and hence $\mathbf{B}^r_G(S) = V$. We can similarly show that Π_e is compact. Next we show that $\overline{\Pi}_1$ is compact. Observe that if $(G,k) \in \Pi_1$ and $(G,k+1) \notin \Pi_1$ then we know that we have a set $\mathcal{Z}$ of k vertex disjoint subgraphs in G where each of them is isomorphic to a subgraph in $\mathcal{S}$. Take S as the union of all the vertices appearing in any of the subgraph in $\mathcal{Z}$. Note that $|S| \le s \cdot k$. Observe that S hits all the subgraphs isomorphic to a subgraph in $\mathcal{S}$ and hence $\mathbf{B}_G^r(S) = V$, where $r = O(s)$. This implies that $\overline{\Pi}_1$ is compact. We can similarly show that $\overline{\Pi}_2$ is compact.

It is well known that Π_v and Π_e are p-MIN-CMSO problems while Π_1 and Π_2 are p-MAX-CMSO problems. The proof that Π_v and Π_1 are finite integer index is similar to the proof given for VERTEX-$\mathcal{H}$-COVERING and VERTEX-$\mathcal{H}$-PACKING, where $\mathcal{H}$ is a finite set of connected planar graphs, have finite integer index in Lemma 8.1.20. The only thing we need to replace is minors by subgraphs in that proof. ■

Domination and its Variants

In the r-DOMINATING SET problem, we are given a graph $G = (V,E)$, and a positive integer k, and the objective is to find a subset $S \subseteq V$ such that $B_G^r(S) = V$ and $|S| \le k$. For $r = 1$, if we demand that $G[S]$ is connected then we get CONNECTED DOMINATING SET. A problem is called q-THRESHOLD DOMINATING SET if we demand that $B_G^1(S) = V$ and for all $v \in (V \setminus S)$, $|N(v) \cap S| \ge q$. An independent set C of vertices in a graph $G = (V,E)$ is an efficient dominating set (or perfect code) when each vertex not in C is adjacent to exactly one vertex in C. In EFFICIENT DOMINATING SET problem we are given a graph $G = (V,E)$ and a positive integer k and the objective is to find an efficient dominating set of size at most k.

Lemma 8.1.23 *r-DOMINATING SET, CONNECTED DOMINATING SET, EFFICIENT DOMINATING SET and q-THRESHOLD DOMINATING SET are compact and have finite integer index.*

Proof. All these problems are compact by definition. To show that these problems have finite integer index, we make use of Lemma 8.1.19. Clearly, they are p-MIN-CMSO problems. We now show that these problems are strongly monotone. We first show it for r-DOMINATING SET. Given a t-boundaried graph $G = (V,E)$, with $\partial(G)$ as its boundary, let $S'' \subseteq V$ be a minimum r-dominating set of G. Take $S = S'' \cup \partial(G)$. Now for any $(G' = (V',E'), S') \in \mathcal{F}_t$ such that $\zeta_G((G',S'))$ is finite we have that $S \cup S'$ is a r-dominating set and $|S| \le \zeta_G((G',S')) + t$. This proves that r-DOMINATING SET is strongly monotone. Similarly, we can show that q-THRESHOLD DOMINATING SET is strongly monotone by taking $S = S'' \cup \partial(G)$ where $S'' \subseteq V$ is a minimum q-threshold dominating set of G. To show that CONNECTED DOMINATING SET is strongly monotone we take $S = S'' \cup \partial(G)$ where

$S'' \subseteq V$ is a union of minimum connected dominating set for each connected components of G.

We now prove that Efficient Dominating Set has finite integer index. Let Π be the Dominating Set problem and Π' be the Efficient Dominating Set problem. The property $P(G)$ that G has an efficient dominating set (of any size) is expressible in CMSO and hence this property is finite state in t-boundaried graphs. As argued in the above paragraph, Dominating Set has finite integer index. Furthermore by a theorem of Bange et al. [14], if a graph G has an efficient dominating set, then the size of the minimum efficient dominating set is equal to the size of the minimum dominating set of the graph. Hence for two t-boundaried graphs G_1, G_2 if G_1 and G_2 are in the same equivalence class of P and $G_1 \equiv_\Pi G_2$ then $G_1 \equiv_{\Pi'} G_2$. Hence Π' has a finite set of representatives. ∎

Problems on Directed Graphs

Our results also apply to problems on directed graphs of bounded genus. In this direction we mention three problems considered in the literature. In Directed Domination [4] we are given a directed graph $D = (V, A)$ and a positive integer k and the objective is to find a subset $S \subseteq V$ of size at most k such that for very vertex $u \in V \setminus S$ there is a vertex $v \in S$ such that $(u, v) \in A$. Independent Directed Domination[1] [83] takes as input a directed graph $D = (V, A)$ and a positive integer k and the objective is to find a subset $S \subseteq V$ of size at most k such that S is an independent set and for every vertex $u \in V \setminus S$ there is a vertex $v \in S$ such that $(u, v) \in A$. In the Minimum Leaf Out-branching [82] problem we are given a directed graph $D = (V, A)$ and a positive integer k and the objective is to find a rooted directed spanning tree (with all arcs directed outwards from the vertices) with at least k internal vertices.

Lemma 8.1.24 Independent Directed Domination *is a* p-min-CMSO *compact problem,* Directed Domination *is compact and has finite integer index and* Π=Minimum Leaf Out-branching *is a* p-max-CMSO *problem and* $\overline{\Pi}$ *is compact.*

Proof. Independent Directed Domination and Directed Domination can easily be seen to be p-min-CMSO problems and by their definition they are compact. Directed Domination can be shown to have finite integer index as follows: given a t-boundaried graph $G = (V, E)$, with $\partial(G)$ as its boundary, let $S'' \subseteq V$ be a minimum directed dominating set of G. Take $S = S'' \cup \partial(G)$. Now for any $(G' = (V', E'), S') \in \mathcal{F}_t$ such that $\zeta_G((G', S'))$ is finite we have that

[1]In literature it is known as "Kernels". We call it differently here to avoid confusion with problem kernels.

$S \cup S'$ is a directed dominating set and $|S| \leq \zeta_G((G', S')) + t$. This proves that Directed Dominating Set is strongly monotone.

Let Π be the Minimum Leaf Out-branching problem. Observe that Π is a p-max-CMSO problem. The set of no instances can be seen to be compact by observing the fact that if $(G, k) \in \Pi$ and $(G, k+1) \notin \Pi$ then G has an out-branching with *exactly* k internal vertices with all other vertices being its leaves. This implies that $\overline{\Pi}$ is compact. ▪

Finally by applying Theorem 8.1.3 together with Lemmata 8.1.19, 8.1.20, 8.1.22, 8.1.23, and 8.1.24 we get the following corollary.

Corollary 8.1.25 *For $g \geq 0$,* Feedback Vertex Set*,* Edge Dominating Set*,* Vertex Cover*,* Dominating Set*,* r-Dominating Set*,* q-Threshold Dominating Set*,* Connected Dominating Set*,* Directed Domination*,* Connected Vertex Cover*,* Minimum-Vertex Feedback Edge Set*,* Induced Matching*,* Minimum Maximal Matching*,* Efficient Dominating Set*,* Independent Set*,* Induced d-Degree Subgraph*,* Min Leaf Spanning Tree*,* Triangle Packing*,* Cycle Packing*,* Cycle Domination*,* Maximum Full-Degree Spanning Tree*,* Vertex-$\mathcal{H}$-Packing*,* Vertex-$\mathcal{H}$-Covering*,* Vertex-$\mathcal{S}$-Covering *and* Vertex-$\mathcal{S}$-Packing *admit a linear kernel on graph of genus at most g.*

Corollary 8.1.25 unifies and generalizes results presented in [4, 5, 22, 23, 28, 69, 79, 80, 87, 104, 112]. By applying Theorem 8.1.1, Corollary 8.1.2, and Lemmata 8.1.20, 8.1.22, 8.1.23, and 8.1.24, we get the following corollary for problems which are not finite integer index.

Corollary 8.1.26 *For $g \geq 0$,* Independent Dominating Set*,* Independent Directed Domination*,* Minimum Leaf Out-branching*,* Edge-$\mathcal{S}$-Covering *and* Edge-$\mathcal{S}$-Packing *admit a polynomial kernels on graphs of genus at most g.*

8.1.5 Practical considerations:

Our meta-theorems provide a simple criteria to decide whether a problem admits a polynomial or linear kernel. Our proofs are constructive and essentially provide *meta-algorithms* that construct kernels for problems in an automated way. Of course, it is expected that for concrete problems, tailor-made kernels will give better bounds. Indeed, many of the published proofs for concrete problems have much smaller constant factors than would follow from a direct application of our proofs. However, our approach might be useful for computer aided design of kernelization algorithms: with a Myhill-Nerode approach, we can get a set of rules that transform each region to the size of some minimum representative, and we can let the computer calculate such sizes or even output the reductions of the corresponding kernel. This seems an interesting (and far from trivial) algorithm-engineering problem.

In general, finding linear kernels *with small constant factors* for concrete problems on planar graphs or graphs with small genus remains a worthy topic of further research.

Compendium of Problems Considered in Section 8.1

Problems that have Finite Integer Index and Quasi-Compactness Property – *Linear Kernels* Dominating Set, r-Dominating Set, q-Threshold Dominating Set, Cycle Domination, Vertex Cover, Feedback Vertex Set, Connected Dominating Set, Connected r-Dominating Set, Connected Vertex Cover, Minimum-Vertex Feedback Edge Set, Edge Dominating Set, Minimum Maximal Matching, Efficient Dominating Set, Vertex-$\mathcal{H}$-Covering, Vertex-$\mathcal{S}$-Covering, Almost-Outerplanar, Clique-Transversal.

Problems that have Finite Integer Index and No Instances having Quasi-Compactness Property – *Linear Kernels* Independent Set, Induced d-Degree Subgraph, Min Leaf Spanning Tree, Induced Matching, Triangle Packing, Cycle Packing, Maximum Full-Degree Spanning Tree, Vertex-$\mathcal{H}$-Packing, Vertex-$\mathcal{S}$-Packing.

Problems that are not covered above and are p-min/eq/max-CMSO problems having Compactness Property – *Polynomial Kernels* Independent Dominating Set, Independent Directed Domination, Minimum Leaf Out-branching, Edge-$\mathcal{S}$-Covering, Edge-$\mathcal{S}$-Packing.

Problems that have Finite Integer Index but neither the problem nor its *no*-instances satisfy Quasi-Compactness Property Minimum Partition Into Cliques, Hamiltonin Path Completion.

Problems that do not have Finite Integer Index Longest Path, Longest Cycle, Maximum Cut, Minimum Covering by Cliques, Independent Dominating Set, Minimum Leaf Out-branching.

8.2 Turing Kernels

As we have seen in Chapter 5, it is not always possible to obtain polynomial kernels for all FPT problems. However, the traditional notion of kernelization does not seem to capture all feasible variants of polynomial time preprocessing.

For instance, the results from Chapter 5 can not be used to rule out a "cheat kernelization algorithm" that reduces the input instance (I, k) to $|I|$ independent kernels of size $O(k)$. That is, such an algorithm would output instances $(X_1, k') \dots (X_{|I|}, k')$ such that (I, k) is a *yes* instance if and only if (X_i, k') is a *yes* instance for some i, and if $|X_i| = O(k)$ for every i. While an algorithm like this is not as useful as an actual kernelization algorithm, it could still turn out to be quite helpful in practice. In particular problems admitting such a "cheat kernel" would be very amenable for parallel algorithms. In 2008, Guo and Fellows [16] asked whether there exist problems that admit such "cheat" polynomial kernels, but provably do not admit polynomial a kernel unless PH=Σ_p^3. The results in this section, together with the results in Section 9.1, give an affirmative answer to this question.

We now formally define the notion of *Turing kernelization*. Our notion is somewhat more general than the "cheat kernel" notion proposed by Guo and Fellows in [16]. The reason we find this notion more appropriate is that even though our notion is more general, problems that admit turing kernels have almost as good algorithmic properties as the problems that admit "cheat kernels". In order to define turing kernels we first define the notion of a *t-oracle*.

Definition 8.2.1 *A t-oracle for a parameterized problem* Π *is an oracle that takes as input* (I, k) *with* $|I| \leq t$, $k \leq t$ *and decides whether* $(I, k) \in \Pi$ *in constant time.*

Definition 8.2.2 *A parameterized problem* Π *is said to have* $g(k)$-sized turing kernel *if there is an algorithm which given an input* (I, k) *together with a* $g(k)$*-oracle for* Π *decides whether* $(I, k) \in \Pi$ *in time polynomial in* $|I|$ *and* k.

Observe that the traditional notion of kernelization is a special case of turing kernelization. In particular, a kernel is equivalent to a turing kernel where the kernelization algorithm is only allowed to make one oracle call and must return the same answer as the oracle.

8.2.1 The Rooted k-Leaf Out-Branching Problem

The k-Leaf Out-Branching problem is to find an out-branching, that is a rooted oriented spanning tree, with at least k leaves in a given digraph. The problem has recently received much attention from the viewpoint of parameterized algorithms [9, 8, 26, 94, 40]. Here, we take a kernelization based approach to the k-Leaf-Out-Branching problem. We give the first polynomial kernel for Rooted k-Leaf-Out-Branching, a variant of k-Leaf-Out-Branching where the root of the tree searched for is also a part of the input. Our kernel has cubic size and is obtained using extremal combinatorics. Our kernel for Rooted k-Leaf-Out-Branching yields a "cheat kernel" for the

k-LEAF-OUT-BRANCHING problem. In Section 9.1 we show that the k-LEAF-OUT-BRANCHING problem does not admit a polynomial kernel unless PH=Σ_p^3. These two results put together dempnstrate that turing kernelization indeed is more powerful than kernelization in the traditional sense. We now give all the data reduction rules we apply on the given instance of ROOTED k-LEAF OUT-BRANCHING to shrink its size.

Proposition 8.2.3 ([94]) *Let D be a digraph and r be a vertex from which every vertex in $V(D)$ is reachable. Then if we have an out-tree rooted at r with k leaves then we also have an out-branching rooted at r with k leaves.*

RedRule 8.2.1.1 [Reachability Rule] *If there exists a vertex u which is disconnected from the root r, then return* NO.

For the ROOTED k-LEAF OUT-TREE problem, Rule 8.2.1.1 translates into following: If a vertex u is disconnected from the root r, then remove u and all in-arcs to u and out-arcs from u.

RedRule 8.2.1.2 [Useless Arc Rule] *If vertex u disconnects a vertex v from the root r, then remove the arc vu.*

Lemma 8.2.4 *Reduction Rules 8.2.1.1 and 8.2.1.2 are correct.*

Proof. If there exists a vertex which can not be reached from the root r then a digraph can not have any r-out-branching. For Reduction Rule 8.2.1.2, all paths from r to v contain the vertex u and thus the arc vu is a back arc in any r-out-branching of D. ■

RedRule 8.2.1.3 [Bridge Rule] *If an arc uv disconnects at least two vertices from the root r, contract arc uv.*

Lemma 8.2.5 *Reduction Rule 8.2.1.3 is correct.*

Proof. Let the arc uv disconnect at least two vertices v and w from r and let D' be the digraph obtained from D by contracting the arc uv. Let T be an r-out-branching of D with at least k leaves. Since every path from r to w contains the arc uv, T contains uv as well and neither u nor v are leaves of T. Let T' be the tree obtained from T by contracting uv. T' is an r-out-branching of D' with at least k leaves.

In the opposite direction, let T' be an r-out-branching of D' with at least k leaves. Let u' be the vertex in D' obtained by contracting the arc uv, and let x be the parent of u' in T'. Notice that the arc xu' in T' was initially the arc xu before the contraction of uv, since there is no path from r to v avoiding u in D. We make

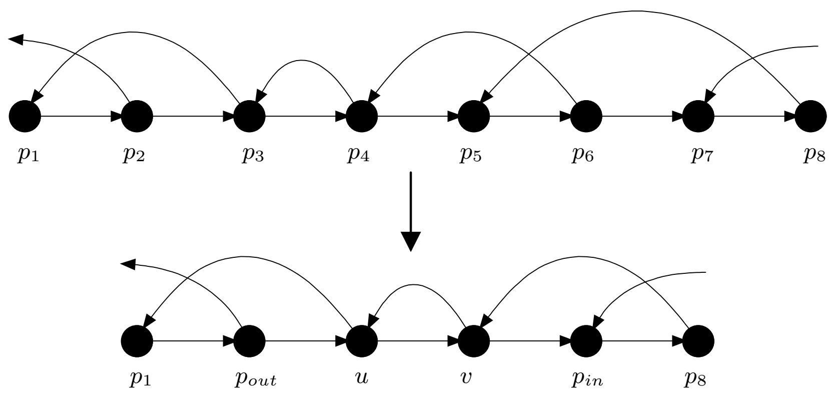

Figure 8.1: An Illustration of Reduction Rule 8.2.1.5.

an r-out-branching T of D from T', by replacing the vertex u' by the vertices u and v and adding the arcs xu, uv and arc sets $\{vy : u'y \in A(T') \wedge vy \in A(D)\}$ and $\{uy : u'y \in A(T') \wedge vy \notin A(D)\}$. All these arcs belong to $A(D)$ because all out-neighbors of u' in D' are out-neighbors either of u or of v in D. Finally, u' must be an inner vertex of T' since u' disconnects w from r. Hence T has at least as many leaves as T'. ∎

RedRule 8.2.1.4 [Avoidable Arc Rule] *If a vertex set S, $|S| \leq 2$, disconnects a vertex v from the root r, $vw \in A(D)$ and $xw \in A(D)$ for all $x \in S$, then delete the arc vw.*

Lemma 8.2.6 *Reduction Rule 8.2.1.4 is correct.*

Proof. Let D' be the graph obtained by removing the arc vw from D and let T be an r-out-branching of D. If $vw \notin A(T)$, T is an r-out-branching of D', so suppose $vw \in A(T)$. Any r-out-branching of D contains the vertex v, and since all paths from r to v contain some vertex $x \in S$, some vertex $u \in S$ is an ancestor of v in T. Let $T' = (T \cup uw) \setminus vw$. T' is an out-branching of D'. Furthermore, since u is an ancestor of v in T, T' has at least as many leaves as T. For the opposite direction observe that any r-out-branching of D' is also an r-out-branching of D. ∎

RedRule 8.2.1.5 [Two Directional Path Rule] *If there is a path $P = p_1p_2 \dots p_{l-1}p_l$ with $l = 7$ or $l = 8$ such that*

- *p_1 and $p_{in} \in \{p_{l-1}, p_l\}$ are the only vertices with in-arcs from the outside of P.*
- *p_l and $p_{out} \in \{p_1, p_2\}$ are the only vertices with out-arcs to the outside of P.*

- *The path P is the unique out-branching of $D[V(P)]$ rooted at p_1.*
- *There is a path Q that is the unique out-branching of $D[V(P)]$ rooted at p_{in}.*
- *The vertex after p_{out} on P is not the same as the vertex after p_l on Q.*

Then delete $R = P \setminus \{p_1, p_{in}, p_{out}, p_l\}$ and all arcs incident to these vertices from D. Add two vertices u and v and the arc set $\{p_{out}u, uv, vp_{in}, p_lv, vu, up_1\}$ to D.

Notice that every vertex on P has in-degree at most 2 and out-degree at most 2. Figure 8.1 gives an example of an application of Reduction Rule 8.2.1.5.

Lemma 8.2.7 *Reduction Rule 8.2.1.5 is correct.*

Proof. Let D' be the graph obtained by performing Reduction Rule 8.2.1.5 to a path P in D. Let P_u be the path $p_1p_{out}uvp_{in}p_l$ and Q_v be the path $p_{in}p_lvup_1p_{out}$. Notice that P_u is the unique out-branching of $D'[V(P_u)]$ rooted at p_1 and that Q_v is the unique out-branching of $D'[V(P_u)]$ rooted at p_{in}.

Let T be an r-out-branching of D with at least k leaves. Notice that since P is the unique out-branching of $D[V(P)]$ rooted at p_1, Q is the unique out-branching of $D[V(P)]$ rooted at p_{in} and p_1 and p_{in} are the only vertices with in-arcs from the outside of P, $T[V(P)]$ is either a path or the union of two vertex disjoint paths. Thus, T has at most two leaves in $V(P)$ and at least one of the following three cases must apply.

1. $T[V(P)]$ is the path P from p_1 to p_l.
2. $T[V(P)]$ is the path Q from p_{in} to p_{out}.
3. $T[V(P)]$ is the vertex disjoint union of a path $\tilde{P}$ that is a subpath of P rooted at p_1, and a path $\tilde{Q}$ that is a subpath of Q rooted at p_{in}.

In the first case we can replace the path P in T by the path P_u to get an r-out-branching of D' with at least k leaves. Similarly, in the second case, we can replace the path Q in T by the path Q_v to get an r-out-branching of D' with at least k leaves. For the third case, observe that $\tilde{P}$ must contain p_{out} since $p_{out} = p_1$ or p_1 appears before p_{out} on Q and thus, p_{out} can only be reached from p_1. Similarly, $\tilde{Q}$ must contain p_l. Thus, $T \setminus R$ is an r-out-branching of $D \setminus R$. We build an r-out-branching T' of D' by taking $T \setminus R$ and letting u be the child of p_{out} and v be the child of p_l. In this case T and T' have same number of leaves outside of $V(P)$ and T has at most two leaves in $V(P)$ while both u and v are leaves in T'. Hence T' has at least k leaves.

To show the other direction, let T' be an r-out-branching of D' with at least k leaves. Notice that since P_u is the unique out-branching of $D'[V(P_u)]$ rooted at p_1, Q_v is the unique out-branching of $D'[V(P_u)]$ rooted at p_{in} and p_1 and p_{in} are the only vertices with in-arcs from the outside of $V(P_u)$, $T'[V(P_u)]$ is either a path or the union of two vertex disjoint paths. Thus, T' has at most two leaves in $V(P_u)$ and at least one of the following three cases must apply.

1. $T'[V(P_u)]$ is the path P_u from p_1 to p_l.
2. $T'[V(P_u)]$ is the path Q_v from p_{in} to p_{out}.
3. $T'[V(P_u)]$ is the vertex disjoint union of a path $\tilde{P}_u$ that is a subpath of P_u rooted at p_1, and a path $\tilde{Q}_v$ that is a subpath of Q_v rooted at p_{in}.

In the first case we can replace the path P_u in T' by the path P to get an r-out-branching of D with at least k leaves. Similarly, in the second case, we can replace the path Q_v in T' by the path Q to get an r-out-branching of D' with at least k leaves. For the third case, observe that $\tilde{P}_u$ must contain p_{out} since $p_{out} = p_1$ or p_1 appears before p_{out} on Q_v and thus, p_{out} can only be reached from p_1. Similarly, $\tilde{Q}_v$ must contain p_l. Thus, $T' \setminus \{u, v\}$ is an r-out-branching of $D' \setminus \{u, v\}$. Let x be the vertex after p_{out} on P, and let y be the vertex after p_l on Q. Vertices x and y must be distinct vertices in R and thus there must be two vertex disjoint paths P_x and Q_y rooted at x and y, respectively, so that $V(P_x) \cup V(Q_y) = R$. We build an r-out-branching T from $(T' \setminus \{u, v\}) \cup P_x \cup Q_y$ by letting x be the child of p_{out} and y be the child of p_{in}. In this case T' and T have the same number of leaves outside of $V(P)$ and T' has at most two leaves in $V(P_u)$ while both the leaf of P_u and the leaf of Q_v are leaves in T. Hence T has at least k leaves. ∎

We say that a digraph D is a *reduced instance* of Rooted k-Leaf Out-Branching if none of the reduction rules (Rules 1–5) can be applied to D. It is easy to observe from the description of the reduction rules that we can apply them in polynomial time, resulting in the following lemma.

Lemma 8.2.8 *For a digraph D on n vertices, we can obtain a reduced instance D' in polynomial time.*

8.2.2 Bounding Entry and Exit Points

We show that any reduced *no*-instance of Rooted k-Leaf Out-Branching must have at most $O(k^3)$ vertices. In order to do so we start with T, a breadth-first search-tree (or BFS-tree for short) rooted at r, of a reduced instance D and look at a path P of T such that every vertex on P has out-degree one in T. We now bound the number of endpoints of arcs with one endpoint in P and one endpoint outside of P.

Let D be a reduced *no*-instance, and T be a BFS-tree rooted at r. The BFS-tree T has at most $k-1$ leaves and hence at most $k-2$ vertices with out-degree at least 2 in T. Now, let $P = p_1 p_2 \ldots p_l$ be a path in T such that all vertices in $V(P)$ have out-degree 1 in T (P does not need to be a maximal path of T). Let T_1 be the subtree of T induced by the vertices reachable from r in T without using vertices in P and let T_2 be the subtree of T rooted at the child r_2 of p_l in T. Since T is a BFS-tree, it does not have any forward arcs, and thus $p_l r_2$ is the only arc from P to T_2. Thus all arcs originating in P and ending outside of P must have their endpoint in T_1.

Lemma 8.2.9 *Let D be a reduced instance, T be a BFS-tree rooted at r, and $P = p_1 p_2 \dots p_l$ be a path in T such that all vertices in $V(P)$ have out-degree* 1 *in T. Let $up_i \in A(D)$, for some i between* 1 *and l, be an arc with $u \notin P$. There is a path P_{up_i} from r to p_i using the arc up_i, such that $V(P_{up_i}) \cap V(P) \subseteq \{p_i, p_l\}$.*

Proof. Let T_1 be the subtree of T induced by the vertices reachable from r in T without using vertices in P and let T_2 be the subtree of T rooted at the child r_2 of p_l in T. If $u \in V(T_1)$ there is a path from r to u avoiding P. Appending the arc up_i to this path yields the desired path P_{up_i}, so assume $u \in V(T_2)$. If all paths from r to u use the arc $p_{l-1}p_l$ then $p_{l-1}p_l$ is an arc disconnecting p_l and r_2 from r, contradicting the fact that Reduction Rule 8.2.1.3 can not be applied. Let P' be a path from r to u not using the arc $p_{l-1}p_l$. Let x be the last vertex from T_1 visited by P'. Since P' avoids $p_{l-1}p_l$ we know that P' does not visit any vertices of $P \setminus \{p_l\}$ after x. We obtain the desired path P_{up_i} by taking the path from r to x in T_1 followed by the subpath of P' from x to u appended by the arc up_i. ■

Corollary 8.2.10 *Let D be a reduced* no-*instance, T be a BFS-tree rooted at r and $P = p_1 p_2 \dots p_l$ be a path in T such that all vertices in $V(P)$ have out-degree* 1 *in T. There are at most k vertices in P that are endpoints of arcs originating outside of P.*

Proof. Let S be the set of vertices in $P \setminus \{p_l\}$ that are endpoints of arcs originating outside of P. For the sake of contradiction suppose that there are at least $k+1$ vertices in P that are endpoints of arcs originating outside of P. Then $|S| \geq k$. By Lemma 8.2.9 there exists a path from the root r to every vertex in S, that avoids vertices of $P \setminus \{p_l\}$ as an intermediate vertex. Using these paths we can build an r-out-tree with every vertex in S as a leaf. This r-out-tree can be extended to a r-out-branching with at least k leaves by Proposition 8.2.3, contradicting the fact that D is a *no*-instance. ■

Lemma 8.2.11 *Let D be a reduced* no-*instance, T be a BFS-tree rooted at r and $P = p_1 p_2 \dots p_l$ be a path in T such that all vertices in $V(P)$ have out-degree* 1 *in T. There are at most* $7(k-1)$ *vertices outside of P that are endpoints of arcs originating in P.*

Proof. Let X be the set of vertices outside P which are out-neighbors of the vertices on P. Let P' be the path from r to p_1 in T and r_2 be the unique child of p_l in T. First, observe that since there are no forward arcs, r_2 is the only out-neighbor of vertices in $V(P)$ in the subtree of T rooted at r_2. In order to bound the size of X, we differentiate between two kinds of out-neighbors of vertices on P.

- Out-neighbors of P that are not in $V(P')$.
- Out-neighbors of P in $V(P')$.

First, observe that $|X \setminus V(P')| \leq k-1$. Otherwise we could have made an r-out-tree with at least k leaves by taking the path $P'P$ and adding $X \setminus V(P')$ as leaves with parents in $V(P)$.

In the rest of the proof we bound $|X \cap V(P')|$. Let Y be the set of vertices on P' with out-degree at least 2 in T and let $P_1, P_2, \ldots, P_t$ be the remaining subpaths of P' when vertices in Y are removed. For every $i \leq t$, $P_i = v_{i1}v_{i2}\ldots v_{iq}$. We define the vertex set Z to contain the two last vertices of each path P_i. The number of vertices with out-degree at least 2 in T is upper bounded by $k-2$ as T has at most $k-1$ leaves. Hence, $|Y| \leq k-2$, $t \leq k-1$ and $|Z| \leq 2(k-1)$.

Claim 8.2.12 *For every path $P_i = v_{i1}v_{i2}\ldots v_{iq}$, $1 \leq i \leq t$, there is either an arc u_iv_{iq-1} or u_iv_{iq} where $u_i \notin V(P_i)$.*

To see the claim observe that the removal of arc $v_{iq-2}v_{iq-1}$ does not disconnect the root r from both v_{iq-1} and v_{iq} else Rule 8.2.1.3 would have been applicable to our reduced instance. For brevity assume that v_{iq-1} is reachable from r after the removal of arc $v_{iq-2}v_{iq-1}$. Hence there exists a path from r to v_{iq}. Let u_iv_{iq} be the last arc of this path. The fact that the BFS-tree T does not have any forward arcs implies that $u_i \notin V(P_i)$.

To every path $P_i = v_{i1}v_{i2}\ldots v_{iq}$, $1 \leq i \leq t$, we associate an interval $I_i = v_{i1}v_{i2}\ldots v_{iq-2}$ and an arc $u_iv_{iq'}$, $q' \in \{q-1, q\}$. This arc exists by Claim 8.2.12. Claim 8.2.12 and Lemma 8.2.9 together imply that for every path P_i there is a path P_{ri} from the root r to $v_{iq'}$ that does not use any vertex in $V(P_i) \setminus \{v_{iq-1}, v_{iq}\}$ as an intermediate vertex. That is, $V(P_{ri} \cap (V(P_i) \setminus \{v_{iq-1}, v_{iq}\}) = \emptyset$.

Let P'_{ri} be a subpath of P_{ri} starting at a vertex x_i before v_{i1} on P' and ending in a vertex y_i after v_{iq-2} on P'. We say that a path P'_{ri} *covers* a vertex x if x is on the subpath of P' between x_i and y_i and we say that it *covers* an interval I_j if x_i appears before v_{j1} on the path P' and y_i appears after v_{jq-2} on P'. Observe that the path P'_{ri} covers the interval I_i.

Let $\mathcal{P} = \{P'_1, P'_2, \ldots, P'_l\} \subseteq \{P'_{r1}, \ldots, P'_{rt}\}$ be a minimum collection of paths, such that every interval I_i, $1 \leq i \leq t$, is covered by at least one of the paths in $\mathcal{P}$. Furthermore, let the paths of $\mathcal{P}$ be numbered by the appearance of their first vertex on P'. The minimality of $\mathcal{P}$ implies that for every $P'_i \in \mathcal{P}$ there is an interval $I'_i \in \{I_1, \ldots, I_t\}$ such that P'_i is the only path in $\mathcal{P}$ that covers I'_i.

Claim 8.2.13 *For every $1 \leq i \leq l$, no vertex of P' is covered by both P'_i and P'_{i+3}.*

The path P'_{i+1} is the only path in $\mathcal{P}$ that covers the interval I'_{i+1} and hence P'_i does not cover the last vertex of I'_{i+1}. Similarly P'_{i+2} is the only path in $\mathcal{P}$ that covers the interval I'_{i+2} and hence P'_{i+3} does not cover the first vertex of I'_{i+2}. Thus the set of vertices covered by both P'_i and P'_{i+3} is empty.

Since paths P'_i and P'_{i+3} do not cover a common vertex, we have that the end vertex of P'_i appears before the start vertex of P'_{i+3} on P' or is the same as the start vertex of P'_{i+3}. Partition the paths of $\mathcal{P}$ into three sets $\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2$, where path $P'_i \in \mathcal{P}_{i\, mod\, 3}$. Also let $\mathcal{I}_i$ be the set of intervals covered by $\mathcal{P}_i$. Observe that every interval I_j, $1 \le j \le t$, is part of some $\mathcal{I}_i$ for $i \in \{0,1,2\}$.

Let $i \le 3$ and consider an interval $I_j \in \mathcal{I}_i$. There is a path $P_{j'} \in \mathcal{P}_i$ that covers I_j such that both endpoints of $P_{j'}$ and none of the inner vertices of $P_{j'}$ lie on P'. Furthermore for any pair of paths P_a, $P_b \in \mathcal{P}_i$ such that $a < b$, there is a subpath in P' from the endpoint of P_a to the starting point of P_b. Thus for every $i \le 3$ there is a path P^*_i from the root r to p_1 which does not use any vertex of the intervals covered by the paths in $\mathcal{P}_i$.

We now claim that the total number of vertices on intervals I_j, $1 \le j \le t$, which are out-neighbors of vertices on $V(P)$ is bounded by $3(k-1)$. If not, then for some i, the number of out-neighbors in $\mathcal{I}_i$ is at least k. Now we can make an r-out-tree with k leaves by taking any r-out-tree in $D[V(P^*_i) \cup V(P)]$ and adding the out-neighbors of the vertices on $V(P)$ in $\mathcal{I}_i$ as leaves with parents in $V(P)$.

Summing up the obtained upper bounds yields $|X| \le (k-1) + |\{r_2\}| + |Y| + |Z| + 3(k-1) \le (k-1) + 1 + (k-2) + 2(k-1) + 3(k-1) = 7(k-1)$, concluding the proof. ■

Remark: Observe that the path P used in Lemmas 8.2.9 and 8.2.11 and Corollary 8.2.10 need not be a maximal path in T with its vertices having out-degree one in T.

8.2.3 Bounding the Length of a Path

Now we bound the size of any maximal path with every vertex having out-degree one in T and use the results proved here together with the ones in the precious section to bound the size of any reduced *no*-instance of ROOTED k-LEAF OUT-BRANCHING by $O(k^3)$. For a reduced instance D, a BFS-tree T of D rooted at r, let $P = p_1p_2 \dots p_l$ be a path in T such that all vertices in $V(P)$ have out-degree 1 in T, and let S be the set of vertices in $V(P) \setminus \{p_l\}$ with an in-arc from the outside of P.

Definition 8.2.14 *A subforest $F = (V(P), A(F))$ of $D[V(P)]$ is said to be a* nice forest of P *if the following three properties are satisfied: (a) F is a forest of directed trees rooted at vertices in S; (b) If $p_ip_j \in A(F)$ and $i < j$ then p_i has out-degree at least 2 in F or p_j has in-degree 1 in D; and (c) If $p_ip_j \in A(F)$ and $i > j$ then for all $q > i$, $p_qp_j \notin A(D)$.*

In order to bound the size of a reduced *no*-instance D we are going to consider a nice forest with the maximum number of leaves. However, in order to do this, we first need to show the existence of a nice forest of P.

In the following discussion let D be a reduced *no*-instance, T be a BFS-tree T of D rooted at r, $P = p_1p_2 \dots p_l$ be a path in T such that all vertices in $V(P)$ have out-degree 1 in T and S be the set of vertices in $V(P) \setminus \{p_l\}$ with an in-arc from the outside of P.

Lemma 8.2.15 *There is a nice forest in P.*

Proof. We define a subgraph F of $D[V(P)]$ as follows. The vertex set of F is $V(P)$ and an arc p_tp_s is in $A(F)$ if $p_s \notin S$ and t is the largest number so that $p_tp_s \in A(D)$. Notice that all arcs of F are covered by property (b) in the definition of a nice forest.

We prove that F is a forest. Suppose for contradiction that there is a cycle C in F. By definition of F every vertex has in-degree at most 1, C must be a directed cycle. Since every vertex in S has in-degree 0 in F, $C \cap S = \emptyset$. Consider the highest numbered vertex p_i on C. Since P has no forward arcs, p_{i-1} is the predecessor of p_i in C. The construction of F implies that there can not be an arc p_qp_i where $q > i$ in $A(D)$. Also, p_i does not have any in-arcs from outside of P. Thus, p_{i-1} disconnects p_i from the root. Hence, by Rule 8.2.1.2 $p_ip_{i-1} \notin A(D)$. Let p_j be the predecessor of p_{i-1} in C. Then $j < i-1$, since $p_ip_{i-1} \notin A(D)$ and p_i is the highest numbered vertex in C. Hence $j = i-2$. This contradicts the fact that D is a reduced instance since the arc $p_{i-2}p_{i-1}$ disconnects p_{i-1} and p_i from the root r implying that Rule 8.2.1.3 can be applied. Since F is a forest and since every vertex in $V(P)$ except for vertices in S have in-degree 1 we conclude that F is a forest of directed trees rooted at vertices in S. Since F is a forest and P has no forward arcs, F is a nice forest. ∎

For a nice forest F of P, we define the set of *key* vertices of F to be the set of vertices in S, the leaves of F, the vertices of F with out-degree at least 2 and the set of vertices whose parent in F has out-degree at least 2.

Lemma 8.2.16 *Let F be a nice forest of P. There are at most* $5(k-1)$ *key vertices of F.*

Proof. By the proof of Corollary 8.2.10 there is an r-out-tree T_S with $(V(T_S) \cap V(P)) \subseteq (S \cup \{p_l\})$ and $(A(T_S) \cap A(P)) = \emptyset$, such that all vertices in $S \setminus \{p_l\}$ are leaves of T_S. We build an r-out-tree $T_F = (V(T_S) \cup V(P), A(T_S) \cup A(F))$. Notice that every leaf of F is a leaf of T_F, except possibly for p_l. Since D is a *no*-instance T_F has at most $k-1$ leaves and $k-2$ vertices with out-degree at least 2. Thus, F has at most k leaves and at most $k-2$ vertices with out-degree at least 2. Hence the number of vertices in F whose parent in F has out-degree at least 2 is at most $2k-2$. Finally, by Corollary 8.2.10, $|S| \leq k$. Adding up these upper bounds yields that there are at most $k-1+k-2+2k-2+k = 5(k-1)$ key vertices of F. ∎

We can now turn our attention to a nice forest F of P with the maximum number of leaves. Our goal is to show that if the key vertices of F are too spaced out on P then some of our reduction rules must apply. First, however, we need some more observations about the interplay between P and F.

Observation 8.2.17 [Unique Path] *For any two vertices p_i, p_j in $V(P)$ such that $i < j$, $p_ip_{i+1} \dots p_j$ is the only path from p_i to p_j in $D[V(P)]$.*

Proof. As T is a BFS-tree it has no forward arcs. So any vertex set $X = \{p_1, p_2, \dots, p_q\}$ with $q < |V(P)|$, the arc p_qp_{q+1} is the only arc in D from a vertex in X to a vertex in $V(P) \setminus X$. ∎

Corollary 8.2.18 *No arc p_ip_{i+1} is a forward arc of F.*

Proof. If p_ip_{i+1} is a forward arc of F then there is a path from p_i to p_{i+1} in F. By Observation 8.2.17 p_ip_{i+1} is the unique path from p_i to p_{i+1} in $D[V(P)]$. Hence $p_ip_{i+1} \in A(F)$ contradicting the fact that it is a forward arc. ∎

Observation 8.2.19 *Let p_tp_j be an arc in $A(F)$ such that neither p_t nor p_j are key vertices, and $t \in \{j-1, j+1, \dots, l\}$. Then for all $q > t$, $p_qp_j \notin A(D)$.*

Observation 8.2.19 follows directly from the definitions of a nice forest and key vertices.

Observation 8.2.20 *If neither p_i nor p_{i+1} are key vertices, then either $p_ip_{i+1} \notin A(F)$ or $p_{i+1}p_{i+2} \notin A(F)$.*

Proof. Assume for contradiction that $p_ip_{i+1} \in A(F)$ and $p_{i+1}p_{i+2} \in A(F)$. Since neither p_i nor p_{i+1} are key vertices, both p_{i+1} and p_{i+2} must have in-degree 1 in D. Then the arc p_ip_{i+1} disconnects both p_{i+1} and p_{i+2} from the root r and Rule 8.2.1.3 can be applied, contradicting the fact that D is a reduced instance. ∎

In the following discussion let F be a nice forest of P with the maximum number of leaves and let $P' = p_xp_{x+1} \dots p_y$ be a subpath of P containing no key vertices, and additionally having the property that $p_{x-1}p_x \notin A(F)$ and $p_yp_{y+1} \notin A(F)$.

Lemma 8.2.21 *$V(P')$ induces a directed path in F.*

Proof. We first prove that for any arc $p_ip_{i+1} \in A(P')$ such that $p_ip_{i+1} \notin A(F)$, there is a path from p_{i+1} to p_i in F. Suppose for contradiction that there is no path from p_{i+1} to p_i in F, and let x be the parent of p_{i+1} in F. Then p_ip_{i+1} is not a backward arc of F and hence $F' = (F \setminus xp_{i+1}) \cup \{p_ip_{i+1}\}$ is a forest of out-trees rooted at vertices in S. Also, since p_{i+1} is not a key vertex, x has out-degree 1 in

F and thus x is a leaf in F'. Since p_i is not a leaf in F, F' has one more leaf than F. Now, every vertex with out-degree at least 2 in F has out-degree at least 2 in F'. Additionally, p_i has out-degree 2 in F'. Hence F' is a nice forest of P with more leaves than F, contradicting the choice of F.

Now, notice that by Observation 8.2.17 any path in $D[V(P)]$ from a vertex $u \in V(P')$ to a vertex $v \in V(P')$ that contains a vertex $w \notin V(P')$ must contain either the arc $p_{x-1}p_x$ or the arc $p_y p_{y+1}$. Since neither of those two arcs are arcs of F it follows that for any arc $p_i p_{i+1} \in A(P')$ such that $p_i p_{i+1} \notin A(F)$, there is a path from p_{i+1} to p_i in $F[V(P')]$. Hence $F[V(P')]$ is weakly connected, that is, the underlying undirected graph is connected. Since every vertex in $V(P')$ has in-degree 1 and out-degree 1 in F we conclude that $F[V(P')]$ is a directed path. ■

In the following discussion let Q' be the directed path $F[V(P')]$.

Observation 8.2.22 *For any pair of vertices $p_i, p_j \in V(P')$ if $i \leq j-2$ then p_j appears before p_i in Q'.*

Proof. Suppose for contradiction that p_i appears before p_j in Q'. By Observation 8.2.17 $p_i p_{i+1} p_{i+2} \dots p_j$ is the unique path from p_i to p_j in $D[V(P')]$. This path contains both the arc $p_i p_{i+1}$ and $p_{i+1} p_{i+2}$ contradicting Observation 8.2.20. ■

Lemma 8.2.23 *All arcs of $D[V(P')]$ are contained in $A(P') \cup A(F)$.*

Proof. Since P has no forward arcs it is enough to prove that any arc $p_j p_i \in A(D[V(P')])$ with $i < j$ is an arc of F. Suppose this is not the case and let p_q be the parent of p_i in F. We know that p_i has in-degree at least 2 in D and also since p_i is not a key vertex p_q has in-degree one in F. Hence by definition of F being a nice forest, we have that for every $t > q$, $p_t p_i \notin A(D)$. It follows that $i < j < q$. By Lemma 8.2.21 $F[V(P')]$ is a directed path Q' containing both p_i and p_j. If p_j appears after p_i in Q', Observation 8.2.22 implies that $i = j-1$ and that p_j has in-degree 1 in D since F is a nice forest. Thus p_i separates p_j from the root and Rule 8.2.1.2 can be applied to $p_j p_i$ contradicting the fact that D is a reduced instance. Hence p_j appears before p_i in Q'.

Since p_j is an ancestor of p_i in F and p_q is the parent of p_i in F, p_j is an ancestor of p_q in F and hence $p_q \in V(Q') = V(P')$. Now, p_j comes before p_q in Q' and $j < q$ so Observation 8.2.22 implies that $q = j+1$ and that p_q has in-degree 1 in D since F is a nice forest. Thus p_j separates p_q from the root r and both $p_j p_i$ and $p_q p_i$ are arcs of D. Hence Rule 8.2.1.4 can be applied to remove the arc $p_q p_i$ contradicting the fact that D is a reduced instance. ■

Lemma 8.2.24 *If $|P'| \geq 3$ there are exactly 2 vertices in P' that are endpoints of arcs starting outside of P'.*

Proof. By Observation 8.2.17, $p_{x-1}p_x$ is the only arc between $\{p_1, p_2, \ldots, p_{x-1}\}$ and P'. By Lemma 8.2.21, $F[V(P')]$ is a directed path Q'. Let p_q be the first vertex on Q' and notice that the parent of p_q in F is outside of $V(P')$. Observation 8.2.22 implies that $q \geq y-1$. Hence p_q and p_x are two distinct vertices that are endpoints of arcs starting outside of P'. It remains to prove that they are the only such vertices. Let p_i be any vertex in $P' \setminus \{p_x, p_q\}$. By Lemma 8.2.21 $V(P')$ induces a directed path Q' in F, and since p_q is the first vertex of Q', the parent of p_i in F is in $V(P')$. Observation 8.2.19 yields then that $p_t p_i \notin A(D)$ for any $t > y$. ■

Observation 8.2.25 *Let $Q' = F[V(P')]$. For any pair of vertices u, v such that there is a path $Q'[uv]$ from u to v in Q', $Q'[uv]$ is the unique path from u to v in $D[V(P')]$.*

Proof. By Lemma 8.2.21 Q' is a directed path $f_1 f_2 \ldots f_{|P'|}$ and let $Q'[f_1 f_i]$ be the path $f_1 f_2 \ldots f_i$. We prove that for any $i < |Q'|$, $f_i f_{i+1}$ is the only arc from $V(Q'[f_1 f_i])$ to $V(Q'[f_{i+1} f_{|P'|}])$. By Lemma 8.2.23 all arcs of $D[V(P')]$ are either arcs of P' or arcs of Q'. Since Q' is a path, $f_i f_{i+1}$ is the only arc from $V(Q'[f_1 f_i])$ to $V(Q'[f_{i+1} f_{|P'|}])$ in Q'. By Corollary 8.2.18 there are no arcs from $V(Q'[f_1 f_i])$ to $V(Q'[f_{i+1} f_{|P'|}])$ in P', except possibly for $f_i f_{i+1}$. ■

Lemma 8.2.26 *For any vertex $x \notin V(P')$ there are at most 2 vertices in P' with arcs to x.*

Proof. Suppose there are 3 vertices p_a, p_b, p_c in $V(P')$ such that $a < b < c$ and such that $p_a x, p_b x, p_c x \in A(D)$. By Lemma 8.2.21 $Q' = F[V(P')]$ is a directed path. If p_a appears before p_b in Q' then Observation 8.2.22 implies that $a+1 = b$ and that p_b has in-degree 1 in D. Then p_a separates p_b from the root and hence Rule 8.2.1.4 can be applied to remove the arc $p_b x$ contradicting the fact that D is a reduced instance. Hence p_b appears before p_a in Q'. By an identical argument p_c appears before p_b in Q'.

Let P_b be a path in D from the root to p_b and let u be the last vertex in P_b outside of $V(P')$. Let v be the vertex in P_b after u. By Lemma 8.2.24, u is either p_x or the first vertex p_q of Q'. If $u = p_x$ then Observation 8.2.17 implies that P_b contains p_a, whereas if $u = p_q$ then Observation 8.2.25 implies that P_b contains p_c. Thus the set $\{p_a, p_c\}$ separates p_b from the root and hence Rule 8.2.1.4 can be applied to remove the arc $p_b x$ contradicting the fact that D is a reduced instance. ■

Corollary 8.2.27 *There are at most $14(k-1)$ vertices in P' with out-neighbors outside of P'.*

Proof. By Lemma 8.2.11 there are at most $7(k-1)$ vertices that are endpoints of arcs originating in P'. By Lemma 8.2.26 each such vertex is the endpoint of at most 2 arcs from vertices in P'. ∎

Lemma 8.2.28 $|P'| \leq 154(k-1)+10$.

Proof. Assume for contradiction that $|P'| > 154(k-1)+10$ and let X be the set of vertices in P' with arcs to vertices outside of P'. By Corollary 8.2.27, $|X| \leq 14(k-1)$. Hence there is a subpath of P' on at least $(154(k-1)+10)/(14(k-1)+1) = 9$ vertices containing no vertices of X. By Observation 8.2.20 there is a subpath $P'' = p_a p_{a+1} \dots p_b$ of P' on 7 or 8 vertices such that neither $p_{a-1}p_a$ nor $p_b p_{b+1}$ are arcs of F. By Lemma 8.2.21 $F[V(P'')]$ is a directed path Q''. Let p_q and p_t be the first and last vertices of Q'', respectively. By Lemma 8.2.24 p_a and p_q are the only vertices with in-arcs from outside of P''. By Observation 8.2.22 $p_q \in \{p_{b-1}, p_b\}$ and $p_t \in \{p_a, p_{a+1}\}$. By the choice of P'' no vertex of P'' has an arc to a vertex outside of P'. Furthermore, since P'' is a subpath of P' and Q'' is a subpath of Q' Lemma 8.2.23 implies that p_b and p_t are the only vertices of P' with out-arcs to the outside of P''. By Lemma 8.2.17, the path P'' is the unique out-branching of $D[V(P'')]$ rooted at p_a. By Lemma 8.2.25, the path Q'' is the unique out-branching of $D[V(P'')]$ rooted at p_q. By Observation 8.2.22 p_{b-2} appears before p_{a+2} in Q'' and hence the vertex after p_b in Q'' and p_{t+1} is not the same vertex. Thus Rule 8.2.1.5 can be applied on P'', contradicting the fact that D is a reduced instance. ∎

Lemma 8.2.29 *Let D be a reduced* no-*instance to* Rooted k-Leaf Out-Branching. *Then $|V(D)| = O(k^3)$. More specifically, $|V(D)| \leq 1540k^3$.*

Proof. Let T be a BFS-tree of D. T has at most $k-1$ leaves and at most $k-2$ inner vertices with out-degree at least 2. The remaining vertices can be partitioned into at most $2k-3$ paths $P_1 \dots P_t$ with all vertices having out-degree 1 in T. We prove that for every $q \in \{1, \dots, t\}$, $|P_q| = O(k^2)$. Let F be a nice forest of P_q with the maximum number of leaves. By Lemma 8.2.16, F has at most $5(k-1)$ key vertices. Let p_i and p_j be consecutive key vertices of F on P_q. By Observation 8.2.20, there is a path $P' = p_x p_{x+1} \dots p_y$ containing no key vertices, with $x \leq i+1$ and $y \geq j-1$, such that neither $p_{x-1}p_x$ nor $p_y p_{y+1}$ are arcs of F. By Lemma 8.2.28 $|P'| \leq 154(k-1)+10$ so $|P_q| \leq (5(k-1)+1)(154(k-1)+10)+3(5(k-1))$. Hence, $|V(D)| \leq 2k(5k(154(k-1)+10+3)) \leq 1540k^3 = O(k^3)$. ∎

Lemma 8.2.29 results in a cubic kernel for Rooted k-Leaf Out-Branching as follows.

Theorem 8.2.30 Rooted k-Leaf Out-Branching *and* Rooted k-Leaf Out-Tree *admits a kernel of size $O(k^3)$.*

Proof. Let D be the reduced instance of Rooted k-Leaf Out-Branching obtained in polynomial time using Lemma 8.2.8. If the size of D is more than $1540k^3$ then return Yes. Else we have an instance of size bounded by $O(k^3)$. The correctness of this step follows from Lemma 8.2.29 which shows that any reduced *no*-instance to Rooted k-Leaf Out-Branching has size bounded by $O(k^3)$. The result for Rooted k-Leaf Out-Tree follows similarly. ■

Chapter 9

Kernelization Lower Bounds

9.1 Polynomial Parameter Transformations with Turing Kernels

In the previous chapter we gave a cubic kernel for Rooted k-Leaf Out-Branching, yielding a polynomial turing kernel for k-Leaf Out-Branching. It is natural to ask whether k-Leaf Out-Branching has a polynomial kernel. The answer to this question, somewhat surprisingly, is no, unless PH=Σ_p^3. To show this we first show that k-Leaf Out-Tree admits no polynomial kernel unless PH=Σ_p^3 by giving a composition algorithm. Then we give a polynomial parameter transformation from k-Leaf Out-Tree to k-Leaf Out-Branching. Our polynomial parameter transformation differs somewhat from the common way a Karp reduction is done. In particular, to give the transformation we employ the cubic kernel for Rooted k-Leaf Out-Branching that we proved in the previous chapter together with the fact that NP-completeness guarantees the existence of polynomial time reductions.

Theorem 9.1.1 k-Leaf Out-Tree *has no polynomial kernel unless* PH=Σ_p^3.

Proof. The problem is NP-complete [9]. We prove that it is compositional and thus, Theorem 5.1.6 will imply the statement of the theorem. A simple composition algorithm for this problem is as follows. On input $(D_1, k), (D_2, k), \ldots, (D_t, k)$ output the instance (D, k) where D is the disjoint union of $D_1, \ldots, D_t$. Since an out-tree must be completely contained in a connected component of the underlying undirected graph of D, (D, k) is a yes-instance to k-Leaf Out-Tree if and only if any out of (D_i, k), $1 \leq i \leq t$, is. This concludes the proof. ■

A *willow* graph [53] $D = (V, A_1 \cup A_2)$ is a directed graph such that $D' = (V, A_1)$ is a directed path $P = p_1 p_2 \ldots p_n$ on all vertices of D and $D'' = (V, A_2)$ is a directed acyclic graph with one vertex r of in-degree 0, such that every arc of A_2 is a backwards arc of P. p_1 is called the *bottom* vertex of the willow, p_n

is called the *top* of the willow and P is called the *stem*. A *nice willow* graph $D = (V, A_1 \cup A_2)$ is a willow graph where $p_n p_{n-1}$ and $p_n p_{n-2}$ are arcs of D, neither p_{n-1} nor p_{n-2} are incident to any other arcs of A_2 and $D'' = (V, A_2)$ has a p_n-out-branching.

Observation 9.1.2 *Let $D = (V, A_1 \cup A_2)$ be a nice willow graph. Every out-branching of D with the maximum number of leaves is rooted at the top vertex p_n.*

Proof. Let $P = p_1 p_2 \dots p_n$ be the stem of D and suppose for contradiction that there is an out-branching T with the maximum number of leaves rooted at p_i, $i < n$. Since D is a nice willow $D' = (V, A_2)$ has a p_n-out-branching T'. Since every arc of A_2 is a back arc of P, $T'[\{v_j : j \geq i\}]$ is an p_n-out-branching of $D[\{v_j : j \geq i\}]$. Then $T'' = (V, \{v_x v_y \in A(T') : y \geq i\} \cup \{v_x v_y \in A(T) : y < i\})$ is an out-branching of D. If $i = n - 1$ then p_n is not a leaf of T since the only arcs going out of the set $\{p_n, p_{n-1}\}$ start in p_n. Thus, in this case, all leaves of T are leaves of T'' and p_{n-1} is a leaf of T'' and not a leaf of T, contradicting the fact that T has the maximum number of leaves. ∎

Lemma 9.1.3 k-LEAF OUT-TREE *in nice willow graphs is NP-complete under Karp reductions.*

Proof. We reduce from the well known NP-complete SET COVER problem [72]. A *set cover* of a universe U is a family $\mathcal{F}'$ of sets over U such that every element of u appears in some set in $\mathcal{F}'$. In the SET COVER problem one is given a family $\mathcal{F} = \{S_1, S_2, \dots S_m\}$ of sets over a universe U, $|U| = n$, together with a number $b \leq m$ and asked whether there is a set cover $\mathcal{F}' \subset \mathcal{F}$ with $|\mathcal{F}'| \leq b$ of U. In our reduction we will assume that every element of U is contained in at least one set in $\mathcal{F}$. We will also assume that $b \leq m - 2$. These assumptions are safe because if either of them does not hold, the SET COVER instance can be resolved in polynomial time. From an instance of SET COVER we build a digraph $D = (V, A_1 \cup A_2)$ as follows. The vertex set V of D is a root r, vertices s_i for each $1 \leq i \leq m$ representing the sets in $\mathcal{F}$, vertices e_i, $1 \leq i \leq n$ representing elements in U and finally 2 vertices p and p'.

The arc set A_2 is as follows, there is an arc from r to each vertex s_i, $1 \leq i \leq m$ and there is an arc from a vertex s_i representing a set to a vertex e_j representing an element if $e_j \in S_i$. Furthermore, rp and rp' are arcs in A_2. Finally, we let $A_1 = \{e_{i+1} e_i : 1 \leq i < n\} \cup \{s_{i+1} s_i : 1 \leq i < m\} \cup \{e_1 s_m, s_1 p, pp', p'r\}$. This concludes the description of D. We now proceed to prove that there is a set cover $\mathcal{F}' \subset \mathcal{F}$ with $|\mathcal{F}'| \leq b$ if and only if there is an out-branching in D with at least $n + m + 2 - b$ leaves.

Suppose that there is a set cover $\mathcal{F}' \subset \mathcal{F}$ with $|\mathcal{F}'| \leq b$. We build a directed tree T rooted at r as follows. Every vertex s_i, $1 \leq i \leq m$, p and p' has r as their parent. For every element e_j, $1 \leq i \leq n$ we chose the parent of e_j to be s_i such that $e_j \in S_i$ and $S_i \in \mathcal{F}'$ and for every $i' < i$ either $S_{i'} \notin |\mathcal{F}'|$ or $e_j \notin S_{i'}$. Since the only inner nodes of T except for the root r are vertices representing sets in the set cover, T is an out-branching of D with at least $n + m + 2 - b$ leaves.

In the other direction suppose that there is an out-branching T of D with at least $n + m + 2 - b$ leaves, and suppose that T has the most leaves out of all out-branchings of D. Since D is a nice willow with r as top vertex, Observation 9.1.2 implies that T is an r-out-branching of D. Now, if there is an arc $e_{i+1}e_i \in A(T)$ then let s_j be a vertex such that $e_i \in S_j$. Then $T' = (T \setminus e_{i+1}e_i) \cup s_je_i$ is an r-out-branching of D with as many leaves as T. Hence, without loss of generality, for every i between 1 and n, the parent of e_i in T is some s_j. Let $\mathcal{F}' = \{S_i : s_i \text{ is an inner vertex of } T\}$. $\mathcal{F}'$ is a set cover of U with size at most $n + m + 2 - (n + m + 2 - b) = b$, concluding the proof. ∎

Theorem 9.1.4 k-Leaf Out-Branching *does not admit a polynomial kernel unless* PH=Σ_p^3.

Proof. We give a polynomial parameter transformation from k-Leaf Out-Tree to k-Leaf Out-Branching. Let (D, k) be an instance to k-Leaf Out-Tree. For every vertex $v \in V$ we make an instance (D, v, k) to Rooted k-Leaf Out-Tree. Clearly, (D, k) is a yes-instance for k-Leaf Out-Tree if and only if (D, v, k) is a yes-instance to Rooted k-Leaf Out-Tree for some $v \in V$. By Theorem 8.2.30 Rooted k-Leaf Out-Tree has a $O(k^3)$ kernel, so we can apply the kernelization algorithm for Rooted k-Leaf Out-Tree separately to each of the n instances of Rooted k-Leaf Out-Tree to get n instances (D_1, v_1, k), (D_2, v_2, k), …, (D_n, v_n, k) with $|V(D_i)| = O(k^3)$ for each $i \leq n$. By Lemma 9.1.3, k-Leaf Out-Branching in nice willow graphs is NP-complete under Karp reductions, so we can reduce each instance (D_i, v_i, k) of Rooted k-Leaf Out-Tree to an instance (W_i, b_i) of k-Leaf Out-Branching in nice willow graphs in polynomial time in $|D_i|$, and hence in polynomial time in k. Thus, in each such instance, $b_i \leq (k + 1)^c$ for some fixed constant c independent of both n and k. Let $b_{max} = \max_{i \leq n} b_i$. Without loss of generality, $b_i = b_{max}$ for every i. This assumption is safe because if it does not hold we can modify the instance (W_i, b_i) by replacing b_i with b_{max}, subdividing the last arc of the stem $b_{max} - b_i$ times and adding an edge from r_i to each subdivision vertex.

From the instances (W_1, b_{max}), …, (W_n, b_{max}) we build an instance $(D', b_{max} + 1)$ of k-Leaf Out-Branching. Let r_i and s_i be the top and bottom vertices of W_i, respectively. We build D' simply by taking the disjoint union of the willows graphs $W_1, W_2, \ldots, W_n$ and adding in an arc r_is_{i+1} for $i < n$ and the arc r_ns_1. Let C be the directed cycle in D obtained by taking the stem of D' and adding the arc r_ns_1.

If for any $i \leq n$, W_i has an out-branching with at least b_{max} leaves, then W_i has an out-branching rooted at r_i with at least b_{max} leaves. We can extend this to an out-branching of D' with at least $b_{max} + 1$ leaves by following C from r_i. In the other direction suppose D' has an out-branching T with at least $b_{max} + 1$ leaves. Let i be the integer such that the root r of T is in $V(W_i)$. For any vertex v in $V(D')$ outside of $V(W_i)$, the only path from r to v in D' is the directed path from r to v in C. Hence, T has at most 1 leaf outside of $V(W_i)$. Thus, $T[V(W_1)]$ contains an out-tree with at least b_{max} leaves. Since $b_{max} \leq (k+1)^c$ the reduction is a polynomial parameter transformation from k-Leaf Out-Tree to k-Leaf Out-Branching. By Theorem 9.1.1 and Proposition 5.2.2 k-Leaf Out-Branching has no polynomial kernel unless PH=Σ_p^3. ∎

9.2 Explicit Identification in Composition Algorithms

The kernelization lower bound for k-Leaf Out-Branching presented in the previous section, together with a result of Bodlaender et al. [24] that the Disjoint Cycles problem does not admit a polynomial kernel were the first non-trivial applications of the framework developed by Bodlaender et al. [20] and Fortnow and Santhanam [70]. In this section we give several non-trivial applications of this framework. In particular we describe a "cookbook" for showing kernelization lower bounds using explicit identification. We then apply this cookbook to show that a wide variety of problems do not admit polynomial kernels, resolving several open problems posed in the literature [10, 15, 78, 81, 111]. To show that a problem does not admit a polynomial size kernel we go through the following steps.

1. Find a suitable parameterization of the problem considered. Quite often parameterizations that impose extra structure make it easier to give a composition algorithm.

2. Define a suitable colored version of the problem. This is in order to get more control over how solutions to problem instances can look.

3. Show that the colored version of the problem is NP-complete.

4. Give a polynomial parameter transformation from the colored to the uncolored version. This will imply that if the uncolored version has a polynomial kernel then so does the colored version. Hence kernelization lower bounds for the colored version directly transfer to the original problem.

5. Show that the colored version parameterized by k is solvable in time $2^{k^c} \cdot n^{O(1)}$ for a fixed constant c.

6. Finally, show that the colored version is compositional and thus has no polynomial kernel. To do so, proceed as follows.

(a) If the number of instances in the input to the composition algorithm is at least 2^{k^c} then running the parameterized algorithm on each instance takes time polynomial in input size. This automatically yields a composition algorithm.

(b) If the number of instances is less than 2^{k^c}, every instance receives an unique identifier. Notice that in order to uniquely code the identifiers (ID) of all instances, k^c bits per instance is sufficient. The IDs are coded either as an integer, or as a subset of a $poly(k)$ sized set.

(c) Use the coding power provided by colors and IDs to complete the composition algorithm.

In the following sections we show how to apply this approach to show incompressibility and kernelization lower bounds for a variety of problems.

9.2.1 Steiner Tree, Variants of Vertex Cover, and Bounded Rank Set Cover

The problems Steiner Tree, Connected Vertex Cover (ConVC), Capacitated Vertex Cover (CapVC), and Bounded Rank Set Cover are defined as follows. In Steiner Tree we are given a graph a graph $G = (T \cup N, E)$ and an integer k and asked for a vertex set $N' \subseteq N$ of size at most k such that $G[T \cup N']$ is connected. In ConVC we are given a graph $G = (V, E)$ and an integer k and asked for a vertex cover of size at most k that induces a connected subgraph in G. A *vertex cover* is a set $C \subseteq V$ such that each edge in E has at least one endpoint in C. The problem CapVC takes as input a graph $G = (V, E)$, a capacity function $cap : V \to \mathbb{N}^+$ and an integer k, and the task is to find a vertex cover C and a mapping from E to C in such a way that at most $cap(v)$ edges are mapped to every vertex $v \in C$. Finally, an instance of Bounded Rank Set Cover consists of a set family $\mathcal{F}$ over a universe U where every set $S \in \mathcal{F}$ has size at most d, and a positive integer k. The task is to find a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ of size at most k such that $\cup_{S \in \mathcal{F}'} S = U$. All four problems are known to be NP-complete (e.g., see [72] and the proof of Theorem 9.2.3); in this section, we show that the problems do not admit polynomial kernels for the parameter $(|T|, k)$ (in the case of Steiner Tree), k (in the case of ConVC and CapVC), and (d, k) (in the case of Bounded Rank Set Cover), respectively. To this end, we first use the framework presented at the beginning of this chapter to prove that another problem, which is called RBDS, does not have a polynomial kernel. Then, by giving polynomial parameter transformations from RBDS to the above problems, we show the non-existence of polynomial kernels for these problems.

In Red-Blue Dominating Set (RBDS) we are given a bipartite graph $G = (T \cup N, E)$ and an integer k and asked whether there exists a vertex set $N' \subseteq N$ of size at most k such that every vertex in T has at least one neighbor in N'. We show that RBDS parameterized by $(|T|, k)$ does not have a polynomial kernel. In the literature, the sets T and N are called "blue vertices" and "red vertices", respectively. In this paper we will call the vertices "terminals" and "nonterminals" in order to avoid confusion with the colored version of the problem that we are going to introduce. RBDS is equivalent to Set Cover and Hitting Set and is, therefore, NP-complete [72].

In the colored version of RBDS, denoted by Colored Red-Blue Dominating Set (Col-RBDS), the vertices of N are colored with colors chosen from $\{1, \ldots, k\}$, that is, we are additionally given a function $col \colon N \to \{1, \ldots, k\}$, and N' is required to contain exactly one vertex of each color. We will now follow the framework described at the beginning of this chapter.

Lemma 9.2.1 (1) *The unparameterized version of* Col-RBDS *is* NP*-complete.* (2) *There is a polynomial parameter transformation from* Col-RBDS *to* RBDS*.* (3) Col-RBDS *is solvable in* $2^{|T|+k} \cdot |T \cup N|^{O(1)}$ *time.*

Proof. (1) It is easy to see that Col-RBDS is in NP. To prove its NP-hardness, we reduce the NP-complete problem RBDS to Col-RBDS: Given an instance $(G = (T \cup N, E), k)$ of RBDS, we construct an instance $(G' = (T \cup N', E'), k, col)$ of Col-RBDS where the vertex set N' consists of k copies $v^1, \ldots, v^k$ of every vertex $v \in V$, one copy of each color. That is, $N' = \bigcup_{a \in \{1,\ldots,k\}} \{v^a \mid v \in N\}$, and the color of every vertex $v^a \in N_a$ is $col(v^a) = a$. The edge set E' is given by

$$E' = \bigcup_{a \in \{1,\ldots,k\}} \{\{u, v^a\} \mid u \in T \wedge a \in \{1, \ldots, k\} \wedge \{u, v\} \in E\}.$$

Now, there is a set $S \subset N$ of size k dominating all vertices in T in G if and only if in G', there is a set $S' \subset N'$ of size k containing one vertex of each color.

(2) Given an instance $(G = (T \cup N, E), k, col)$ of Col-RBDS, we construct an instance $(G' = (T' \cup N, E'), k)$ of RBDS. The set T' consists of all vertices from T plus k additional vertices $z_1, \ldots, z_k$. The edge set E' consists of all edges from E plus the edges

$$\{\{z_a, v\} \mid a \in \{1, \ldots, k\} \ \wedge \ v \in N \ \wedge \ col(v) = a\}.$$

Now, there is a set $S' \subset N'$ of size k dominating all vertices in T' in G' if and only if in G, there is a set $S \subset N$ of size k containing one vertex of each color.

(3) To solve Col-RBDS in the claimed running time, we first use the reduction given in (2) from Col-RBDS to RBDS. The number $|T'|$ of terminals in the constructed instance of RBDS is $|T| + k$. Next, we transform the RBDS instance (G', k) into an instance $(\mathcal{F}, U, k)$ of Set Cover where the elements in U one-to-one correspond to the vertices in T' and the sets in $\mathcal{F}$

one-to-one correspond to the vertices in N. Since Set Cover can be solved in $O(2^{|U|} \cdot |U| \cdot |\mathcal{F}|)$ time [68, Lemma 2], statement (3) follows.

■

Lemma 9.2.2 Col-RBDS *parameterized by* $(|T|, k)$ *is compositional.*

Proof. Given a sequence

$$(G_1 = (T_1 \cup N_1, E_1), k, col_1), \ldots, (G_t = (T_t \cup N_t, E_t), k, col_t)$$

of Col-RBDS instances with $|T_1| = |T_2| = \ldots = |T_t| = p$, we show how to construct a Col-RBDS instance $(G = (T \cup N, E), k, col)$ as described in Definition 5.1.5.

For $i \in \{1, \ldots, t\}$, let $T_i := \{u_1^i, \ldots, u_p^i\}$ and $N_i := \{v_1^i, \ldots, v_{q_i}^i\}$. We start with adding p vertices $u_1, \ldots, u_p$ to the set T of terminals to be constructed. (We will add more vertices to T later.) Next, we add to the set N of nonterminals all vertices from the vertex sets $N_1, \ldots, N_t$, preserving the colors of the vertices. That is, we set $N = \bigcup_{i \in \{1,\ldots,t\}} N_i$, and for every vertex $v_j^i \in N$ we define $col(v_j^i) = col_i(v_j^i)$. Now, we add the edge set $\bigcup_{i \in \{1,\ldots,t\}} \big\{\{u_{j_1}, v_{j_2}^i\} \mid \{u_{j_1}^i, v_{j_2}^i\} \in E_i\big\}$ to G (see Figure 9.1). The graph G and the coloring col constructed so far have the following property: If at least one of the Col-RBDS instances $(G_1, k, col_1), \ldots, (G_t, k, col_t)$ is a *yes*-instance, then (G, k, col) is also a *yes*-instance because if for any $i \in \{1, \ldots, t\}$ a size-k subset from N_i dominates all vertices in T_i, then the same vertex set selected from N also dominates all vertices in T. However, (G, k, col) may even be a *yes*-instance in the case where all instances $(G_1, k, col_1), \ldots, (G_t, k, col_t)$ are *no*-instances, because in G one can select vertices into the solution that originate from different instances of the input sequence.

To ensure the correctness of the composition, we add more vertices and edges to G. We define for every graph G_i of the input sequence a unique identifier $\mathrm{ID}(G_i)$, which consists of a size-$(p + k)$ subset of $\{1, \ldots, 2(p + k)\}$ Since $\binom{2(p+k)}{p+k} \geq 2^{p+k}$ and since we can assume that the input sequence does not contain more than 2^{p+k} instances, it is always possible to assign unique identifiers to all instances of the input sequence. (Note that if there are more than 2^{p+k} instances, then we can solve all these instances in $\sum_{i=1}^{t} 2^{p+k} \cdot (p + q_i)^{O(1)} \leq t \cdot \sum_{i=1}^{t} (p + q_i)^{O(1)}$ time, which yields a composition algorithm.) For each color pair $(a, b) \in \{1, \ldots, k\} \times \{1, \ldots, k\}$ with $a \neq b$, we add a vertex set $W_{(a,b)} = \{w_1^{(a,b)}, \ldots, w_{2(p+k)}^{(a,b)}\}$ to T, and we add to E the edge set

$$\bigcup_{i \in \{1,\ldots,t\}, j_1 \in \{1,\ldots,q_i\}} \Big\{\{v_{j_1}^i, w_{j_2}^{(a,b)}\} \mid a = col(v_{j_1}^i) \wedge b \in \{1, \ldots, k\} \setminus \{a\} \wedge j_2 \in \mathrm{ID}(G_i)\Big\} \cup$$
$$\bigcup_{i \in \{1,\ldots,t\}, j_1 \in \{1,\ldots,q_i\}} \Big\{\{v_{j_1}^i, w_{j_2}^{(a,b)}\} \mid b = col(v_{j_1}^i) \wedge a \in \{1, \ldots, k\} \setminus \{b\} \wedge j_2 \notin \mathrm{ID}(G_i)\Big\}$$

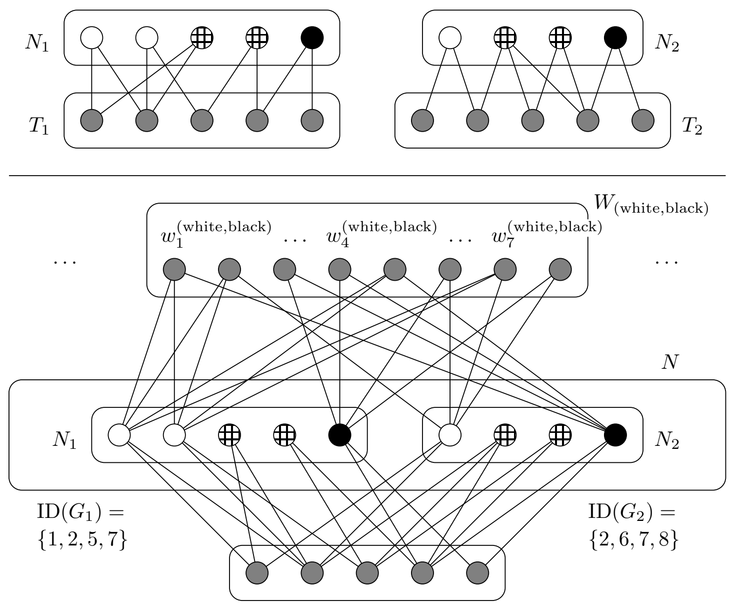
N_1
T_1
N_2
T_2
$W_{(\text{white,black})}$
$w_1^{(\text{white,black})}$ … $w_4^{(\text{white,black})}$ … $w_7^{(\text{white,black})}$
…
…
N
N_1
N_2
ID(G_1) = $\{1,2,5,7\}$
ID(G_2) = $\{2,6,7,8\}$

Figure 9.1: Example for the composition algorithm for COL-RBDS. The upper part of the figure shows an input sequence consisting of two instances with $k = 3$ (there are three colors: white, checkered, and black). The lower part of the figure shows the output of the composition algorithm. For the sake of clarity, only the vertex set $W_{(\text{white,black})}$ is displayed, whereas five other vertex sets $W_{(a,b)}$ with $a, b \in \{\text{white}, \text{checkered}, \text{black}\}$ are omitted. Since $k = 3$ and $p = 5$, each ID should consist of eight numbers, and $W_{(\text{white,black})}$ should contain 16 vertices. For the sake of clarity, the displayed IDs consist of only four numbers each, and $W_{(\text{white,black})}$ contains only eight vertices.

(see Figure 9.1).

Note that the construction conforms to the definition of a composition algorithm; in particular, k remains unchanged and the size of T is polynomial in p, k because $|T| = p + k(k-1) \cdot 2(p+k)$. To prove the correctness of the construction, we show that (G, k, col) has a solution $N' \subseteq N$ if and only if at least one instance (G_i, k, col_i) from the input sequence has a solution $N_i' \subseteq N_i$.

In one direction, if $N_i' \subseteq N_i$ is a solution for (G_i, k, col_i), then the same vertex set chosen from N forms a solution for (G, k, col). To see this, first note that the vertices from T are dominated by the chosen vertices. Moreover, for every color pair $(a, b) \in \{1, \ldots, k\} \times \{1, \ldots, k\}$ with $a \neq b$, each vertex from $W_{(a,b)}$ is either connected to all vertices v from N_i with $col(v) = a$ or to all vertices v from N_i with $col(v) = b$. Since N_i' contains one vertex of each color class from N_i, each vertex in $W_{(a,b)}$ is dominated by a vertex from N chosen into the solution.

In the other direction, to show that any solution $N' \subseteq N$ for (G, k, col) is a solution for at least one instance (G_i, k, col_i), we prove that N' cannot contain vertices originating from different instances of the input sequence. To this end, note that each two vertices in N' must have different colors. Assume, for the sake of a contradiction, that N' contains a vertex $v^{i_1}_{j_1}$ with $col(v^{i_1}_{j_1}) = a$ originating from the instance (G_{i_1}, k, col_{i_1}) and a vertex $v^{i_2}_{j_2}$ with $col(v^{i_2}_{j_2}) = b$ originating from a different instance (G_{i_2}, k, col_{i_2}). Due to the construction of the IDs, we have $\mathrm{ID}(G_{i_1}) \setminus \mathrm{ID}(G_{i_2}) \neq \emptyset$ and $\mathrm{ID}(G_{i_2}) \setminus \mathrm{ID}(G_{i_1}) \neq \emptyset$. This implies that there are vertices in $W_{(a,b)}$ (namely, all vertices $w^{(a,b)}_j$ with $j \in \mathrm{ID}(G_{i_2}) \setminus \mathrm{ID}(G_{i_1})$) and vertices in $W_{(b,a)}$ (namely, all vertices $w^{(b,a)}_j$ with $j \in \mathrm{ID}(G_{i_1}) \setminus \mathrm{ID}(G_{i_2})$) that are neither adjacent to $v^{i_1}_{j_1}$ nor to $v^{i_2}_{j_2}$. Therefore, N' does not dominate all vertices from T, which is a contradiction to the fact that N' is a solution for (G, k, col). ■

Theorem 9.2.3 RED-BLUE DOMINATING SET *and* STEINER TREE*, both parameterized by* $(|T|, k)$*,* CONNECTED VERTEX COVER *and* CAPACITATED VERTEX COVER*, both parameterized by* k*, and* BOUNDED RANK SET COVER*, parameterized by* (k, d)*, do not admit polynomial kernels unless* $PH = \Sigma^3_p$*.*

Proof. For RBDS the statement of the theorem follows directly by Theorem 5.1.6 together with Lemmata 9.2.1 and 9.2.2.

To show that the statement is true for the other four problems, we give polynomial parameter transformations from RBDS to each of them—due to Proposition 5.2.2, this suffices to prove the statement. Let $(G = (T \cup N, E), k)$ be an instance of RBDS. To transform it into an instance $(G' = (T' \cup N, E'), k)$ of STEINER TREE, define $T' = T \cup \{\tilde{u}\}$ where $\tilde{u}$ is a new vertex and let $E' = E \cup \{\{\tilde{u}, v_i\} \mid v_i \in N\}$. It is easy to see that every solution for STEINER TREE on (G', k) one-to-one corresponds to a solution for RBDS on (G, k).

To transform (G, k) into an instance $(G'' = (V'', E''), k'')$ of CONVC, first construct the graph $G' = (T' \cup N, E')$ as described above. The graph G'' is then obtained from G' by attaching a leaf to every vertex in T'. Now, G'' has a connected vertex cover of size $k'' = |T'| + k = |T| + 1 + k$ iff G' has a steiner tree containing k vertices from N iff all vertices from T can be dominated in G by k vertices from N.

Next, we describe how to transform (G, k) into an instance $(G''' = (V''', E'''), cap, k''')$ of CAPVC. First, for each vertex $u_i \in T$, add a clique to G''' that contains four vertices $u^0_i, u^1_i, u^2_i, u^3_i$. Second, for each vertex $v_i \in N$, add a vertex v'''_i to G'''. Finally, for each edge $\{u_i, v_j\} \in E$ with $u_i \in T$ and $v_j \in N$, add the edge $\{u^0_i, v'''_j\}$ to G'''. The capacities of the vertices are defined as follows: For each vertex $u_i \in T$, the vertices $u^1_i, u^2_i, u^3_i \in V'''$ have capacity 1 and the vertex $u^0_i \in V'''$ has capacity $\deg_{G'''}(u^0_i) - 1$. Each vertex v'''_i has capacity $\deg_{G'''}(v'''_i)$. Clearly, in order to cover the edges of the size-4 cliques inserted for the vertices of T, every capacitated vertex cover for G''' must contain all vertices $u^0_i, u^1_i, u^2_i, u^3_i$. Moreover, since the capacity of each vertex u^0_i is too small to cover all edges incident to u^0_i, at least one neighbor v_j''' of u_i^0 must be selected into every capacitated vertex cover for G'''. Therefore, it is not hard to see that G''' has a capacitated vertex cover of size $k''' = 4 \cdot |T| + k$ iff all vertices from T can be dominated in G by k vertices from N.

Finally, to transform (G, k) into an instance $(\mathcal{F}, U, k)$ of Bounded Rank Set Cover, add one element e_i to U for every vertex $u_i \in T$. For every vertex $v_j \in N$, add one set $\{e_i \mid \{u_i, v_j\} \in E\}$ to $\mathcal{F}$. The correctness of the construction is obvious, and since $|U| = |T|$, every set in $\mathcal{F}$ contains at most $d = |T|$ elements. ∎

9.2.2 Unique Coverage

In the Unique Coverage problem we are given a universe U, a family of sets $\mathcal{F}$ over U and an integer k. The problem is to find a sub-family $\mathcal{F}'$ of $\mathcal{F}$ and a set S of elements in U such that $|S| \geq k$ and every element of S appears in exactly one set in $\mathcal{F}'$, that is, the number of elements uniquely covered by $\mathcal{F}'$ is at least k.

In order to obtain our negative results we have to utilize positive kernelization results for the problem. In some sense, we have to compress our instances as much as possible in order to show that what remains is incompressible even though it is big. We utilize the following well-known and simple reduction rules for the problem: (a) If any set $S \in \mathcal{F}$ contains at least k elements, return yes; (b) If any element e is not contained in any set in $\mathcal{F}$, remove e from U; and (c) If none of the above rules can be applied and $|U| \geq k(k-1)$ return yes.

We show that the Unique Coverage problem does not have a polynomial kernel unless PH=Σ_p^3. Notice that while the above reduction rules will compress the instance to an instance with at most $O(k^2)$ elements, this is not a polynomial kernel because there is no polynomial bound on the size of $\mathcal{F}$. Hence, our negative result implies that unless PH=Σ_p^3 the size of $\mathcal{F}$ can not be compressed to a polynomial in k in polynomial time. We start by defining the colorful reduced version Colored Reduced Unique Coverage (Col-Red-UC) of the Unique Coverage problem which is useful for making the composition algorithm. In this version the sets of $\mathcal{F}$ are colored with colors from the set $\{1, \ldots, k\}$ and $\mathcal{F}'$ is required to contain exactly one set of each color. Furthermore, in Col-Red-UC every set S in $\mathcal{F}$ has size at most $k-1$ and $|U| \leq k^2$.

Lemma 9.2.4 (1) *The unparameterized version of* Col-Red-UC *is* NP*-complete.* (2) *There is a polynomial parameter transformation from* Col-Red-UC *to* Unique Coverage. (3) Col-Red-UC *parameterized by k is solvable in time $O(k^{2k^2})$.*

Proof. (1) To show that that Col-Red-UC is NP-complete we reduce from the Unique Coverage problem. For an instance $(\mathcal{F}, U, k)$ of Unique Coverage we first apply the reduction rules above. We now assume that the given instance cannot be reduced any further. Now we make k copies of $\mathcal{F}$, one copy of each color. This new instance has a colored subfamily uniquely covering at least k elements if and only if $(\mathcal{F}, U, k)$ is a yes instance to UNIQUE COVERAGE. Furthermore, even in the new instance we have that $|U| \leq k^2$ and that every set has at most $k-1$ elements.

(2) We now prove that there is a polynomial parameter transformation from COL-RED-UC to UNIQUE COVERAGE. For an instance $(\mathcal{F}, U, k)$ of COL-RED-UC we make a new instance $(\mathcal{H}, U', k')$ to UNIQUE COVERAGE. Let $k' = k(k^2 + 1) + k$ and for every color i we add a set U_i of $k^2 + 1$ new elements to U and make all sets colored with i contain U_i in addition to what they already contain. Thus we have that $U' = U \cup \bigcup_{i \in \{1,\dots,k\}} U_i$. Notice that in order to cover at least $k(k^2 + 1)$ elements uniquely one has to pick exactly one set of each color. This concludes the polynomial parameter transformation.

(3) Finally, observe that COL-RED-UC can be solved in time $O(k^{2k^2})$ because the size of $|U|$ is bounded by k^2. ■

Lemma 9.2.5 *The* COL-RED-UC *problem is compositional.*

Proof. Given a sequence of COL-RED-UC instances $\mathcal{I}_1 = (U, \mathcal{F}_1, k), \dots, \mathcal{I}_t = (U, \mathcal{F}_t, k)$, we construct an COL-RED-UC instance $\mathcal{I} = (U', \mathcal{F}, k')$. If the number of instances t is at least $2^{2k^2 \log k}$ then running the algorithm from Lemma 9.2.4 on all instances takes time polynomial in the input size yielding a trivial composition algorithm. Thus we assume that t is at most $2^{2k^2 \log k}$. We now construct ID's for for every instance, this is done in two steps. In the first step every instance i gets a unique small id $\text{ID}'(\mathcal{I}_i)$ which is a subset of size $k^3/2$ of the set $\{1, \dots, k^3\}$. The identifier of instance i is the set $\text{ID}(\mathcal{I}_i)$ which is defined to be $\text{ID}(\mathcal{I}_i) = \{x \in \mathbb{N} : \lfloor x/k^3 \rfloor \in \text{ID}'(\mathcal{I}_i)\}$. In other words, $\text{ID}(\mathcal{I}_i) = \{k^3 \cdot j + j' \mid j \in \text{ID}'(\mathcal{I}_i) \wedge j' \in \{0, \dots, k^3 - 1\}\}$. Notice that the identifier of every instance is now a subset of size $k^6/2$ of the set $\{1, \dots, k^6\}$ and that the ID's of two different instances differ in at least k^3 places.

We start building the instance $\mathcal{I}$ by letting $U' = U$ and $\mathcal{F} = \mathcal{F}_1 \cup \mathcal{F}_2 \dots \cup \mathcal{F}_t$. The sets have the same color as in their respective instance. For every distinct ordered pair of colors $i, j \leq k$ we add the set $U_{i,j} = \{u_{i,j}^1, \dots, u_{i,j}^{k^6}\}$ to U'. For every instance $\mathcal{I}_p$ we consider the sets colored i and j respectively in $\mathcal{F}_p$. To every set S with color i in $\mathcal{F}_p$ we add the set $\{u_{i,j}^x : x \in \text{ID}(\mathcal{I}_p)\}$. Also, to every set S with color j in $\mathcal{F}_p$ we add the set $\{u_{i,j}^x : x \notin \text{ID}(\mathcal{I}_p)\}$. Finally we set $k' = k(k-1)k^6 + k$. This concludes the construction.

If some $\mathcal{I}_p$ has a colored subfamily $\mathcal{F}'$ covering k elements uniquely, we show that the same subfamily covers k' elements uniquely in $\mathcal{I}$. First note that $\mathcal{F}'$ covers k elements uniquely in U. It remains to prove that for every distinct ordered pair i, j of colors, all elements of $U_{i,j}$ are covered uniquely by $\mathcal{F}'$ in $\mathcal{I}$. Consider an element $u_{i,j}^q \in U_{i,j}$ and let S_i and S_j be the sets colored i and j

respectively in $\mathcal{F}'$. If $q \in \mathrm{ID}(\mathcal{I}_p)$ then S_i contains $u^q_{i,j}$ and S_j does not. Similarly if $q \notin \mathrm{ID}(\mathcal{I}_p)$ then S_j contains $u^q_{i,j}$ and S_i does not. Furthermore no other set of $\mathcal{F}'$ contains $u^q_{i,j}$ and thus this element is uniquely covered.

In the other direction, suppose $\mathcal{I}$ has a colored subfamily $\mathcal{F}'$ covering k' elements uniquely. Suppose for contradiction that there is a color i and a color j such that the set $S_i \in \mathcal{F}'$ with color i and the set $S_j \in \mathcal{F}'$ with color j originate from different instances. Observe that in the set $U_{i,j}$ the sets S_i and S_j intersect in at least k^3 elements, and thus these elements are not covered uniquely by $\mathcal{F}'$. Then the total number of elements that can be uniquely covered by $\mathcal{F}'$ is upper bounded by $k(k-1)k^6 + k^2 - k^3 < k(k-1)k^6 < k'$ yielding a contradiction. Thus all the sets in $\mathcal{F}'$ come from the same instance $\mathcal{I}_p$ and uniquely cover at least k elements in $\mathcal{I}_p$. This concludes the proof. ∎

Theorem 9.2.6 *The* Unique Coverage *problem does not admit a polynomial kernel unless $PH = \Sigma^3_p$.*

9.2.3 Bounded Rank Disjoint Sets

In the Bounded Rank Disjoint Sets problem we are given a family $\mathcal{F}$ over a universe U with every set $S \in \mathcal{F}$ having size at most d together with a positive integer k. The question is whether there exists a subfamily $\mathcal{F}'$ of $\mathcal{F}$ with $|\mathcal{F}'| \geq k$ such that for every pair of sets $S_1, S_2 \in \mathcal{F}'$ we have that $S_1 \cap S_2 = \emptyset$. The problem can be solved in time $2^{O(dk)}n^{O(1)}$ using color-coding and an application of dk-perfect hash families. We outline a proof which shows that this problem does not admit a poly(k,d) kernel. To do so we define a variation of the Perfect Code problem on graphs, which we call Bipartite Regular Perfect Code problem. In Bipartite Regular Perfect Code we are given a bipartite graph $G = (T \cup N, E)$, where every vertex in N has the same degree, and an integer k and asked whether there exists a vertex set $N' \subseteq N$ of size at most k such that every vertex in T has *exactly one neighbor* in N'. The set N' is called a bipartite perfect code. Now we are ready to state the main theorem of this subsection.

Theorem 9.2.7 Bipartite Regular Perfect Code *parameterized by* $(|T|, k)$ *and* Bounded Rank Disjoint Sets *parameterized by* (d, k) *do not have a polynomial kernel unless $PH = \Sigma^3_p$.*

Proof. We can show that Bipartite Regular Perfect Code parameterized by $(|T|, k)$ does not have a polynomial kernel along the lines of the proof of Theorem 9.2.3, which shows that RBDS parameterized by $(|T|, k)$ does not have a polynomial kernel. Bipartite Regular Perfect Code is known to be NP-complete even when every vertex in N has degree exactly 3 [99]. The proof showing that the colored version of Bipartite Regular Perfect Code

(1) is NP-complete, (2) has a fixed-parameter algorithm of the desired kind, and (3) has a polynomial parameter transformation to Bipartite Regular Perfect Code is just a minor modification as for RBDS. For the composition, it is enough to observe that if the input graphs to the composition algorithm are one sided regular then the composed graph will remain one sided regular in Lemma 9.2.2. This is true as every vertex is made adjacent to the same number of newly added vertices and newly added vertices are added to T.

Finally, to show that Bounded Rank Disjoint Sets parameterized by (d, k) does not have a polynomial kernel we give a polynomial parameter transformation from Bipartite Regular Perfect Code to Bounded Rank Disjoint Sets. To this end, given an instance $(G = (T \cup N, E), k)$ for Bipartite Regular Perfect Code, we make an instance $(U, \mathcal{F}, k', d)$ for Bounded Rank Disjoint Sets as follows. Let $U = T$, $\mathcal{F} = \{F_v \mid v \in N,\ F_v = N(v)\}$, $k' = k$ and $d = r$ where r is the degree of any vertex in N. Observe that $k = |T|/r$. From here it easily follows that G has a bipartite perfect code of size k if and only if $(U, \mathcal{F}, k, d)$ has a subfamily $\mathcal{F}'$ of $\mathcal{F}$ of pairwise disjoint sets. ■

9.2.4 Domination and Transversals

In the Small Universe Hitting Set problem we are given a set family $\mathcal{F}$ over a universe U with $|U| \leq d$ together with a positive integer k. The question is whether there exists a subset S in U of size at most k such that every set in $\mathcal{F}$ has a non-empty intersection with S. We show that the Small Universe Hitting Set problem parameterized by the solution size k and the size $d = |U|$ of the universe does not have a kernel of size polynomial in (k, d) unless $PH = \Sigma_p^3$. One should notice that while Hitting Set and Set Cover in fact are the same problem, Small Universe Hitting Set and Small Universe Set Cover are not, because in the former we are restricting the number of potential dominators while in the later we restrict the number of objects to be dominated.

We define the colored version of Small Universe Hitting Set, called Col-SUHS as follows. We are given a set family $\mathcal{F}$ over a universe U with $|U| \leq d$, and a positive integer k. The elements of U are colored with colors from the set $\{1, \ldots, k\}$ and the question is whether there exists a subset $S \subseteq U$ containing exactly one element of each color such that every set in $\mathcal{F}$ has a non-empty intersection with S.

Lemma 9.2.8 (1) *The unparameterized version of* Col-SUHS *is* NP*-complete.* (2) *There is a polynomial parameter transformation from* Col-SUHS *to* Small Universe Hitting Set*.* (3) Col-SUHS *parameterized by d, k is solvable in time $O(2^d \cdot n^{O(1)})$.*

Proof. (1) We show that Col-SUHS is NP-complete by reducing from Small Universe Hitting Set, which is easily seen to be NP-complete by a reduction from Dominating Set. Given an instance $(\mathcal{F}, U, d, k)$ to Small Universe Hitting Set we make an instance $(\mathcal{F}', U', d \cdot k, k)$ of Col-BRHS by letting U' contain k copies of U, with one element of each color. For every set S in $\mathcal{F}$ we make a set S' in $\mathcal{F}'$ by letting S' contain all copies of all elements in S. From the construction it follows that $(\mathcal{F}, U, d, k)$ has a hitting set of size at most k if and only if $(\mathcal{F}', U', d \cdot k, k)$ has a colored hitting set of size k.

(2) We now give a polynomial parameter transformation from Col-SUHS to Small Universe Hitting Set. For an instance $(\mathcal{F}', U, d, k)$ to Col-SUHS we make an instance $(\mathcal{F}, U, d, k)$ to Small Universe Hitting Set as follows. For every i between 1 and k let U_i be the set of elements in U colored with i. We construct $\mathcal{F}$ by starting with $\mathcal{F}'$ and for every i adding the set U_i to $\mathcal{F}'$. Now, a subset S of U hits every set of $\mathcal{F}$ if and only if it hits every set of $\mathcal{F}'$ and contains at least one vertex of each color.

(3) Finally, observe that Col-SUHS parameterized by d, k is solvable in time $O(2^d \cdot n^{O(1)})$ by enumerating all subsets of U and for each set checking whether it is a hitting set with at least one vertex of each color. ■

Lemma 9.2.9 *The problem* Col-SUHS *is compositional.*

Proof. We have to show how, given a sequence of Col-SUHS instances $(\mathcal{F}_1, U, d, k)$, …, $(\mathcal{F}_t, U, d, k)$ to Col-SUHS where $|U| \leq d$, to construct a Col-SUHS instance $(\mathcal{F}, U', d', k')$ as described in Definition 5.1.5.

If the number of instances is at least 2^d then running the algorithm from Lemma 9.2.8 on all instances takes time polynomial in the input size yielding a trivial composition algorithm. Thus we can assume that $t \leq 2^d$. Furthermore, we need the number of instances to be a power of 2. To make this true we add in an appropriate number of no-instances. As we never add more than t extra instances in order to make the number of instances a power of 2 this can be done in polynomial time and hence we can safely assume that t is a power of 2, say 2^l. Observe that since $t \leq 2^d$ we have that $l \leq d$. Now, let every instance be identified by a unique number from 0 to $t-1$.

We let $k' = k+l$ and start building $(\mathcal{F}, U', d', k')$ from $(\mathcal{F}_1, U, d, k), \ldots, (\mathcal{F}_t, U, d, k)$ by letting $U' = U$ and letting elements keep their color. For every $i \leq t$ we add the family $\mathcal{F}_i$ to $\mathcal{F}$. We now add $2l$ new elements $C = \{a_1, b_1, \ldots, a_l, b_l\}$ to U' and for every $i \leq l$, $\{a_i, b_i\}$ comprise a new color class. We conclude the construction by modifying the sets in $\mathcal{F}$ that came from the input instances to the composition algorithm. For every $j \leq t$ we consider all sets in $\mathcal{F}_j$. For every such set S we proceed as follows. Let ID(j) be the identification number of instance number j. For every $i \leq l$ we look at the i'th bit in the binary representation of ID(j). If this bit is set to 1 we add a_i to S and if the bit is set to 0 we add b_i to S. This concludes the construction.

Now, if there is a colored hitting set S for $\mathcal{F}_j$ with $|S| \leq k$ we construct a colored hitting set S' for $\mathcal{F}$ of size $k + l$ as follows. First we add S to S' and then we consider the identification number ID(j) of instance j. For every i between 1 and l we consider the i'th bit of ID(j). If this bit is set to 1 we add b_j to S' else we add a_j to S'. Clearly S' is a hitting set for $\mathcal{F}_i$, has size $k + l$ and contains one vertex of each color. It remains to show that S' hits all other sets of $\mathcal{F}$. Consider any set $X \in \mathcal{F}_q$ for some $q \neq j$. Then there is an i such that ID(q) differs from ID(j) in the i'th bit. If the i'th bit of ID(q) is 1 then the i'th bit of ID(j) is 0 and hence $a_i \in S'$. Since the i'th bit of ID(q) is 1, $a_i \in X$.

In the other direction, suppose there is a colored hitting set S' of size $l + k$ of $\mathcal{F}$. For every $i \leq l$, exactly one out of the vertices a_i and b_i is in S'. Let p be the number between 0 and $2^l - 1$ such that for every i the i'th bit of p is 1 if and only if $b_i \in S'$. Observe that the sets in $\mathcal{F}$ originating from the family $\mathcal{F}_j$ such that ID$(j) = p$ do not contain any of the elements of $S' \cap C$. Thus $S'' = S' \cap U$ is a colored hitting set for $\mathcal{F}_j$ containing at most one element from each color class. S'' can thus be extended to a colored hitting set S of $\mathcal{F}_j$ with $|S| = k$, concluding the proof. ∎

Theorem 9.2.10 Small Universe Hitting Set *parameterized by solution size k and universe size $|U| = d$ does not have a polynomial kernel unless $PH = \Sigma_p^3$. The* Dominating Set *problem parameterized by the solution size k and the size c of a minimum vertex cover of the input graph does not have a polynomial kernel.*

Proof. The first part of the theorem follows immediately from Lemmata 9.2.8 and 9.2.9. To show that the Dominating Set problem parameterized by the solution size k and the size c of a minimum vertex cover of the input graph does not have a polynomial kernel we give a polynomial parameter transformation from Col-SUHS to Dominating Set. On input $(\mathcal{F}, U, d, k)$ to Col-SUHS we make an instance (G, k) to Dominating Set as follows. We start by letting G be the bipartite element-set incidence graph of $(\mathcal{F}, U)$. Now we and add one vertex v_i for every color class i and make v_i adjacent to every vertex in G corresponding to an element of U with color i. This concludes the construction.

First, observe that U is a vertex cover of size d of G. Second, observe that if there is a colored hitting set S with $|S| \leq k$ of $(\mathcal{F}, U, d, k)$ then S is a dominating set of G. Finally, if S is a dominating set of G then for every i there is a vertex in $S \cap N[v_i]$. Hence, $S \cap U$ is a hitting set for $\mathcal{F}$ containing at most one element of each color. Thus this set can be extended to a colorful hitting set of $\mathcal{F}$, concluding the proof. ∎

Theorem 9.2.10 has some interesting consequences. For instance, it implies that the Hitting Set problem parameterized by solution size k and the maximum size d of any set in $\mathcal{F}$ does not have a kernel of size $poly(k, d)$ unless $PH = \Sigma_p^3$. The second part of Theorem 9.2.10 implies that the Dominating Set problem in graphs excluding a fixed graph H as a minor parameterized by

$(k, |H|)$ does not have a kernel of size $poly(k, |H|)$ unless $PH = \Sigma_p^3$. This follows from the well-known fact that every graph with a vertex cover of size c excludes the complete graph K_{c+2} as a minor. Similarly, since every graph with a vertex cover of size c is c-degenerate it follows that the DOMINATING SET problem in d-degenerate graphs does not have a kernel of size $poly(k, d)$ unless $PH = \Sigma_p^3$.

Theorem 9.2.11 *Unless $PH = \Sigma_p^3$ the problems* HITTING SET *parameterized by solution size k and the maximum size d of any set in $\mathcal{F}$,* DOMINATING SET IN H-MINOR FREE GRAPHS *parameterized by $(k, |H|)$, and* DOMINATING SET *parameterized by solution size k and degeneracy d of the input graph do not have a polynomial kernel.*

9.2.5 Numeric Problem: Small Subset Sum

In the SUBSET SUM problem we are given a set S of n integers and a target t and asked whether there is a subset S' of S that adds up to exactly t. In the most common parameterization of this problem one is also given an integer k and asked whether there is a subset S' of S of size at most k that adds up to t. This parameterization, however, is $W[1]$-hard. We consider a stronger parameterization where in addition to k an extra parameter d is provided and the integers in S are required to have size at most 2^d. This version, SMALL SUBSET SUM is trivially fixed parameter tractable by dynamic programming. We believe that SMALL SUBSET SUM is the most restrictive plausible parameterization of the SUBSET SUM problem. We show that even this version does not admit a polynomial kernel, by giving a polynomial parameter transformation from the COLORED RED-BLUE DOMINATING SET (COL-RBDS) problem.

Theorem 9.2.12 SMALL SUBSET SUM *parameterized by (d, k) does not admit a kernel polynomial in (d, k) unless $PH = \Sigma_p^3$.*

Proof. We give a polynomial parameter transformation from the COLORED RED-BLUE DOMINATING SET (COL-RBDS) problem to SMALL SUBSET SUM. Given an instance $(G = (T \cup N, E), k, d)$ to COL-RBDS, such that $|T| = d$ and N has been colored with colors from $\{1, \ldots, k\}$, we build an instance (S, t, k', d') to SMALL SUBSET SUM. For an integer $x \in S$ we treat x both as a number and as a string—the encoding of x in the number system with base $k(k+1)$.

We let t be a length-$(d + 2k)$ string of digits representing the number $1 + k(k+1)/2$ and $k' = k(d+1)$. Now, order the elements of T in some order, say $T = t_1, t_2, \ldots, t_d$. For a vertex $v \in N$ we define the string $Z(v)$ to be a string on d digits, where the i'th digit is set to 1 if $\{v, t_i\} \in E$ and 0 otherwise. For an integer i between 1 and k we define the string $B(i)$ to be a length-k string with zeroes everywhere, except in the i'th digit, which is $1 + k(k+1)/2$. For every vertex $x \in N$ we add a string to S: Let i be the color of x. We add the string

$B(i)Z(x)B(k+1-i)$ to S and we will say that this string (or the number it represents) corresponds to x. Even though the numbers in the Small Subset Sum are uncolored, we color the string corresponding to x with the same color as x in order to ease the discussion. Finally we add a set of uncolored numbers to S: For every i between 1 and d and every j between 1 and k the string $u_{i,j}$ is a string of length $2k+d$ with zeroes everywhere except for the $k+i$'th digit, which is set to j. This concludes the construction. One should notice that every number is bounded by $(k(k+1))^{2k+d}$ and hence we can set $d' = 3(2k+d)\log k$. We prove that there is a set $N' \subseteq N$ containing one vertex of each color such that every vertex of T has a neighbor in N' if and only if there is a set $S' \subseteq S$ of at most k' numbers that add up to t.

Suppose that there is a set $N' \subseteq N$ containing one vertex of each color such that every vertex of T has a neighbor in N'. We pick $S' \subseteq S$ as follows. For every vertex $v \in N'$ we add the string corresponding to v to S'. Furthermore, for every $i \leq d$, let x_i be the number of neighbors the vertex $t_i \in T$ has in N'. We add the set $\{u_{i,j} : 1 \leq j \leq k \wedge j \neq x_i - 1\}$ to S'. Since we picked one number of each color, when we add up the numbers in S' there are no carries. Since every vertex $t_i \in T$ sees at least one and at most k vertices in N', the $k+i$'th digit of the sum of all numbers in S' is exactly $1 + k(k+1)/2$.

In the other direction, suppose there is a set $S' \subseteq S$ with at most k' numbers that add up to t. If S contains no numbers colored 1, then the last digit of $\sum_{x \in S'} x$ must be zero, a contradiction. If S contains at least two numbers colored 1 then $\sum_{x \in S'} x > t$ again yielding a contradiction. Hence S' contains exactly one number colored 1. Let s_1 be the number in S' colored 1. Now, if S' contains no numbers colored 2 then the second last digit of $\sum_{x \in S'} x$ is zero and if S' contains at least 2 numbers colored 2 then $-s_1 + \sum_{x \in S'} x > t - s_1$, again contradicting that $t = \sum_{x \in S'} x$. Repeating the argument for the remaining colors yields that S' contains exactly one number of each color. For every $i \leq k$ we let s_i be the number in S' colored i. For every i let v_i be the vertex in N corresponding to s_i. We prove that for every i the vertex $t_i \in T$ has a neighbor in N'. To achieve this we argue that there is a number s_j such that the $k+i$'th digit of s_j is 1. Suppose for contradiction that this is not the case. Notice that since there is at most one number of each color in S' there are no carries when we add up the numbers of S', even if S' contains all uncolored numbers. Hence, even if S' contains all uncolored numbers whose $k+i$'th digit is non-zero the $k+i$'th digit of $\sum_{x \in S'} x$ must be strictly less than $1 + k(k+1)/2$, yielding the desired contradiction, thereby completing the proof. ∎

Chapter 10

Concluding Remarks and Open Problems

In this thesis, we have provided new methods for showing upper and lower bounds in Parameterized Algorithms and Complexity and in Kernelization. We conclude the thesis with a list of open problems that naturally arise from our results, or to which the techniques developed here might be amenable.

Algorithmic

- Our algorithm for k-Feedback Arc Set in Tournaments runs time $2^{\tilde{O}(\sqrt{k})} + n^{O(1)}$. Can the polylogarithmic term be removed from the exponent, that is, can k-Feedback Arc Set in Tournaments be solved in time $2^{O(\sqrt{k})} + n^{O(1)}$?

- Can the chromatic coding technique be applied to other problems in dense structures? In particular, in the Edge Bipartization problem input is a graph G and integer k. The objective is to find a set of at most k edges whose removal makes the graph bipartite. Is there an $2^{o(k)}$ time algorithm for the Edge Bipartization problem restricted to graphs where every vertex has degree at least $c \cdot n$ for every fixed constant $c > 0$?

Hardness

- Is Max Cut FPT parameterized by cliquewidth? In this problem we are given a graph G of cliquewidth at most k and asked to color the vertices of G black or white such that the maximum number of edges have endpoints with different colors. We conjecture that the problem is W[1]-hard.

- A well-known open problem is whether the Biclique problem is FPT. In this problem we are given a graph G and an integer k and asked whether G contains a complete bipartite graph with both bipartitions of size at least

k as subgraph. This problem is believed to be W[1]-hard. Can a novel application of explicit identification be used to prove that BICLIQUE indeed is W[1]-hard?

Kernelization

- Do all compact p-MIN/EQ/MAX-CMSO problems admit a linear kernel in graphs of bounded genus? If so, do all quasi-compact p-MIN/EQ/MAX-CMSO problems admit a linear kernel in graphs of bounded genus?

- Is there a semantical characterization of the problems that have finite integer index?

- Can Theorems 8.1.1 and 8.1.3 be extended to graph classes excluding a fixed graph H as a minor, or at least for minor closed graph classes with bounded *local treewidth* [66]?

- It does not seem possible to directly apply Theorems 8.1.1 and 8.1.3 to the ODD CYCLE TRANSVERSAL problem. Does ODD CYCLE TRANSVERSAL admit a polynomial kernel in planar graphs?

- LONGEST PATH is known not to admit a polynomial kernel unless PH=Σ_p^3 [20], even when input is restricted to planar graphs. Does LONGEST PATH in planar graphs admit a polynomial size turing kernel?

- FEEDBACK VERTEX SET is known to admit a $O(k^2)$ kernel [132]. Does FEEDBACK VERTEX SET admit a linear size turing kernel?

Kernelization Lower Bounds

- Several of the problems we have considered are parameterized by two variables, like BOUNDED RANK HITTING SET and BOUNDED RANK SET COVER parameterized by solution size k and maximum set size d, and DOMINATING SET IN H-MINOR FREE GRAPHS parameterized by $(k, |H|)$. While we show that the problems do not admit a kernel polynomial in *both* k and d, the problems do admit a polynomial kernel in the solution size k when the other parameter is regarded as a constant. In particular both BOUNDED RANK HITTING SET and BOUNDED RANK SET COVER admit $k^{O(d)}$ kernels while DOMINATING SET IN H-MINOR FREE GRAPHS admits a $k^{f(|H|)}$ kernel. Is there something in between? Or, more specifically, do BOUNDED RANK HITTING SET and BOUNDED RANK SET COVER admit kernels of size $f(d) \cdot k^{O(1)}$, and does DOMINATING SET IN H-MINOR FREE GRAPHS admit a $f(|H|) \cdot k^{O(1)}$ kernel? We coin this kind of kernels, *uniformly polynomial* kernels and ask the general question - which problems admit uniformly polynomial kernels, and which do not?

- Is there a way to rule out the existence of polynomial size turing kernels for an FPT problem, up to some complexity-theoretical assumption?

Bibliography

[1] M. J. Pelsmajer A. V. Kostochka and D. B. West. A list analogue of equitable coloring. *Journal of Graph Theory*, 44:166–177, 2003.

[2] Karl R. Abrahamson and Michael R. Fellows. Finite automata, bounded treewidth and well-quasiordering. In N. Robertson and P. Seymour, editors, *Proceedings of the AMS Summer Workshop on Graph Minors, Graph Structure Theory, Contemporary Mathematics vol. 147*, pages 539–564. American Mathematical Society, 1993.

[3] Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: ranking and clustering. In *ACM Symposium on Theory of Computing (STOC)*, pages 684–693, 2005.

[4] Jochen Alber, Britta Dorn, and Rolf Niedermeier. A general data reduction scheme for domination in graphs. In *SOFSEM '06:*, volume 3831 of *Lecture Notes in Computer Science*, pages 137–147, Berlin, 2006. Springer.

[5] Jochen Alber, Michael R. Fellows, and Rolf Niedermeier. Polynomial-time data reduction for dominating set. *J. ACM*, 51(3):363–384, 2004.

[6] N. Alon. Ranking tournaments. *SIAM J. Discrete Math.*, 20:137–142, 2006.

[7] N. Alon, L. Babai, and A. Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of Algorithms*, 7:567–583, 1986.

[8] Noga Alon, Fedor V. Fomin, Gregory Gutin, Michael Krivelevich, and Saket Saurabh. Better algorithms and bounds for directed maximum leaf problems. In *FSTTCS*, pages 316–327, 2007.

[9] Noga Alon, Fedor V. Fomin, Gregory Gutin, Michael Krivelevich, and Saket Saurabh. Parameterized algorithms for directed maximum leaf problems. In *ICALP*, pages 352–362, 2007.

[10] Noga Alon and Shai Gutner. Kernels for the dominating set problem on graphs with an excluded minor. Technical Report TR08-066, Electronic Colloquium on Computational Complexity (ECCC), 2008.

[11] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. Assoc. Comput. Mach.*, 42(4):844–856, 1995.

[12] Stefan Arnborg, Bruno Courcelle, Andrzej Proskurowski, and Detlef Seese. An algebraic theory of graph reduction. *J. ACM*, 40(5):1134–1164, 1993.

[13] R. Balasubian, Michael R. Fellows, and Venkatesh Raman. An improved fixed-parameter algorithm for vertex cover. *Inf. Process. Lett.*, 65(3):163–168, 1998.

[14] D. W. Bange, A. E. Barkauskas, and P. J. Slater. Efficient dominating sets in graphs. In *Proceedings of 3rd Conference on Discrete Mathematics*, pages 189–199. SIAM, 1988.

[15] Nadja Betzler. Steiner tree problems in the analysis of biological networks. Diploma thesis, Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, Germany, 2006.

[16] H. L. Bodlaender, E. D. Demaine, M. R. Fellows, J. Guo, D. Hermelin, D. Lokshtanov, M. Müller, V. Raman, J. van Rooij, and F. A. Rosamond. Open problems in parameterized and exact computation iwpec 2008. Technical Report UUCS-2008-017, Utrecht University., 2008.

[17] H. L. Bodlaender and D. Kratsch. A note on fixed parameter intractability of some domination-related problems. Private communication, 1994.

[18] Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

[19] Hans L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoret. Comput. Sci.*, 209(1-2):1–45, 1998.

[20] Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. In *Proc. 35th ICALP*, volume 5125 of *LNCS*, pages 563–574. Springer, 2008.

[21] Hans L. Bodlaender and Fedor V. Fomin. Equitable colorings of bounded treewidth graphs. *Theor. Comput. Sci.*, 349(1):22–30, 2005.

[22] Hans L. Bodlaender and Eelko Penninkx. A linear kernel for planar feedback vertex set. In *Proceedings of the Third International Workshop on Parameterized and Exact Computation (IWPEC)*, volume 5018 of *LNCS*, pages 160–171. Springer, 2008.

[23] Hans L. Bodlaender, Eelko Penninkx, and Richard B. Tan. A linear kernel for the k-disjoint cycle problem on planar graphs. In *Proceedings of the 19th International Symposium on Algorithms and Computation (ISAAC)*, volume 5369 of *LNCS*, pages 306–317. Springer, 2008.

[24] Hans L. Bodlaender, Stéphan Thomassé, and Anders Yeo. Analysis of data reduction: Transformations give evidence for non-existence of polynomial kernels. Technical Report UU-CS-2008-030, Department of Information and Computing Sciences, Utrecht University, 2008.

[25] Hans L. Bodlaender and Babette van Antwerpen-de Fluiter. Reduction algorithms for graphs of small treewidth. *Information and Computation*, 167:86–119, 2001.

[26] Paul S. Bonsma and Frederic Dorn. Tight bounds and a fast fpt algorithm for directed max-leaf spanning tree. In *ESA*, pages 222–233, 2008.

[27] Richard B. Borie, R. Gary Parker, and Craig A. Tovey. Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families. *Algorithmica*, 7(5&6), 1992.

[28] Jianer Chen, Henning Fernau, Iyad A. Kanj, and Ge Xia. Parametric duality and kernelization: Lower bounds and upper bounds on kernel size. *SIAM J. Comput.*, 37:1077–1106, 2007.

[29] Jianer Chen, Iyad A. Kanj, and Weijia Jia. Vertex Cover: Further observations and further improvements. *J. Algorithms*, 41(2):280–301, 2001.

[30] Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved parameterized upper bounds for vertex cover. In *MFCS*, pages 238–249, 2006.

[31] Jianer Chen, Yang Liu, Songjian Lu, Barry O'Sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. In *STOC*, pages 177–186, 2008.

[32] Jianer Chen, Songjian Lu, Sing-Hoi Sze, and Fenghui Zhang. Improved algorithms for path, matching, and packing problems. In *SODA*, pages 298–307, 2007.

[33] D. Coppersmith, Lisa Fleischer, and Atri Rudra. Ordering by weighted number of wins gives a good ranking for weighted tournaments. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 776–782, 2006.

[34] Derek G. Corneil and Udi Rotics. On the relationship between clique-width and treewidth. *SIAM J. Comput.*, 34(4):825–847, 2005.

[35] B. Courcelle. The monadic second-order logic of graphs. III. Tree-decompositions, minors and complexity issues. *RAIRO Inform. Théor. Appl.*, 26(3):257–286, 1992.

[36] B. Courcelle, J. A. Makowsky, and U. Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000.

[37] B. Courcelle, J. A. Makowsky, and U. Rotics. On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic. *Discrete Appl. Math.*, 108(1-2):23–52, 2001. International Workshop on Graph-Theoretic Concepts in Computer Science (Smolenice Castle, 1998).

[38] Bruno Courcelle. The monadic second-order logic of graphs I: Recognizable sets of finite graphs. *Information and Computation*, 85:12–75, 1990.

[39] Bruno Courcelle and M. Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theoretical Computer Science*, 109:49–82, 1993.

[40] Jean Daligault, Gregory Gutin, Eun Jung Kim, and Anders Yeo. Fpt algorithms and kernels for the directed k-leaf problem. *CoRR*, abs/0810.4946, 2008.

[41] Babette de Fluiter. *Algorithms for Graphs of Small Treewidth.* PhD thesis, Utrecht University, 1997.

[42] Frank K. H. A. Dehne, Michael R. Fellows, Michael A. Langston, Frances A. Rosamond, and Kim Stevens. An $\mathrm{o}(2^{\mathrm{o}(\mathrm{k})}\mathrm{n}^3)$ fpt algorithm for the undirected feedback vertex set problem. In *COCOON*, pages 859–869, 2005.

[43] Erik D. Demaine, Fedor V. Fomin, Mohammad Taghi Hajiaghayi, and Dimitrios M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and -minor-free graphs. *J. ACM*, 52(6):866–893, 2005.

[44] Erik D. Demaine, Fedor V. Fomin, Mohammadtaghi Hajiaghayi, and Dimitrios M. Thilikos. Fixed-parameter algorithms for (k, r)-center in planar graphs and map graphs. *ACM Trans. Algorithms*, 1(1):33–47, 2005.

[45] Erik D. Demaine and Mohammadtaghi Hajiaghayi. Graphs excluding a fixed minor have grids as large as treewidth, with combinatorial and algorithmic applications through bidimensionality. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA2005*, pages 682–689, 2005.

[46] Erik D. Demaine and MohammadTaghi Hajiaghayi. The bidimensionality theory and its algorithmic applications. *Comput. J.*, 51(3):292–302, 2008.

[47] Erik D. Demaine, Mohammadtaghi Hajiaghayi, and Dimitrios M. Thilikos. The bidimensional theory of bounded-genus graphs. *SIAM J. Discrete Math.*, 20(2):357–371 (electronic), 2006.

[48] Michael Dom, Jiong Guo, Falk Hüffner, Rolf Niedermeier, and Anke Truß. Fixed-parameter tractability results for feedback set problems in tournaments. In *CIAC*, pages 320–331, 2006.

[49] Frederic Dorn, Fedor V. Fomin, and Dimitrios M. Thilikos. Subexponential parameterized algorithms. *Computer Science Review*, 2(1):29–39, 2008.

[50] Rodney G. Downey and Michael R. Fellows. Fixed-parameter intractability. In *Structure in Complexity Theory Conference*, pages 36–49, 1992.

[51] Rodney G. Downey and Michael R. Fellows. Fixed parameter tractability and completeness. In *Complexity Theory: Current Research*, pages 191–225, 1992.

[52] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity.* Springer, 1999.

[53] M. Drescher and A. Vetta. An approximation algorithm for the maximum leaf spanning arborescence problem. *ACM Transactions on Algorithms*, page To appear.

[54] S.E. Dreyfus and R.A. Wagner. The steiner problem in graphs. *Networks*, 1:195–207, 1972.

[55] David Eppstein. Subgraph isomorphism in planar graphs and related problems. *J. Graph Algorithms and Applications*, 3:1–27, 1999.

[56] P. Erdös and J.W.Moon. On sets on consistent arcs in tournaments. *Canadian Mathematical Bulletin*, 8:269–271, 1965.

[57] Wolfgang Espelage, Frank Gurski, and Egon Wanke. Deciding clique-width for graphs of bounded tree-width. *J. Graph Algorithms Appl.*, 7(2):141–180 (electronic), 2003.

[58] Michael Fellows, Daniel Lokshtanov, Neeldhara Misra, Matthias Mnich, Frances Rosamond, and Saket Saurabh. The complexity ecology of parameters: An illustration using bounded max leaf number. Manuscript, 2008.

[59] Michael R. Fellows. On the complexity of vertex set problems. Technical report, Computer Science Department, University of New Mexico, 1988.

[60] Michael R. Fellows, Fedor V. Fomin, Daniel Lokshtanov, Elena Losievskaja, Frances A. Rosamond, and Saket Saurabh. Parameterized low-distortion embeddings - graph metrics into lines and trees. *CoRR*, abs/0804.3028, 2008.

[61] Michael R. Fellows and Michael A. Langston. Nonconstructive advances in polynomial-time complexity. *Inf. Process. Lett.*, 26(3):155–162, 1987.

[62] M.R. Fellows, D. Hermelin, F.A. Rosamond, and S. Vialette. On the parameterized complexity of multiple-interval graph problems. *Theor. Comput. Sci.*, 410(1):53–61, 2009.

[63] M.R. Fellows, D. Lokshtanov, N. Misra, F.A. Rosamond, and S. Saurabh. Graph layout problems parameterized by vertex cover. In *ISAAC*, pages 294–305, 2008.

[64] W. Fernandez de la Vega. On the maximal cardinality of a consistent set of arcs in a random tournament. *J. Combinatorial Theory, Ser. B*, 35:328–332, 1983.

[65] Henning Fernau, Fedor V. Fomin, Daniel Lokshtanov, Daniel Raible, Saket Saurabh, and Yngve Villanger. Kernel(s) for problems with no kernel: On out-trees with many leaves. In *Proc. 26th STACS*, To appear.

[66] J. Flum and M. Grohe. *Parameterized Complexity Theory (Texts in Theoretical Computer Science. An EATCS Series)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[67] F. V. Fomin, S. Saurabh, and D. M. Thilikos. Improving the gap of erdos-posa property for minor-closed graph classes. In *proceedings of 7th Cologne-Twente Workshop on Graphs and Combinatorial Optimization, 2008.* 2008.

[68] Fedor V. Fomin, Dieter Kratsch, and Gerhard J. Woeginger. Exact (exponential) algorithms for the dominating set problem. In *Proc. 30th WG*, volume 3353 of *LNCS*, pages 245–256. Springer, 2004.

[69] Fedor V. Fomin and Dimitrios M. Thilikos. Fast parameterized algorithms for graphs on surfaces: Linear kernel and exponential speed-up. In *Proc. 31st ICALP*, volume 3142 of *LNCS*, pages 581–592. Springer, 2004.

[70] Lance Fortnow and Rahul Santhanam. Infeasibility of instance compression and succinct PCPs for NP. In *Proc. 40th STOC*, pages 133–142. ACM Press, 2008.

[71] András Frank and Éva Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Combinatorica*, 7:49–65, 1987.

[72] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.

[73] Michael U. Gerber and Daniel Kobler. Algorithms for vertex-partitioning problems on graphs with fixed clique-width. *Theoret. Comput. Sci.*, 299(1-3):719–734, 2003.

[74] B. Godlin, T. Kotek, and J.A. Makowsky. Evaluations of graph polynomials. In *WG'08*, LNCS, page to appear. Springer, 2008.

[75] Jens Gramm, Rolf Niedermeier, and Peter Rossmanith. Fixed-parameter algorithms for closest string and related problems. *Algorithmica*, 37(1):25–42, 2003.

[76] J. Guo, H. Moser, and R. Niedermeier. Iterative compression for exactly solving np-hard minimization problems. In *Algorithmics of Large and Complex Networks, Lecture Notes in Computer Science*, page To appear. Springer, 2008.

[77] Jiong Guo, Jens Gramm, Falk Hüffner, Rolf Niedermeier, and Sebastian Wernicke. Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization. *J. Comput. Syst. Sci.*, 72(8):1386–1396, 2006.

[78] Jiong Guo and Rolf Niedermeier. Invitation to data reduction and problem kernelization. *SIGACT News*, 38(1):31–45, 2007.

[79] Jiong Guo and Rolf Niedermeier. Linear problem kernels for NP-hard problems on planar graphs. In *Automata, languages and programming*, volume 4596 of *Lecture Notes in Comput. Sci.*, pages 375–386. Springer, Berlin, 2007.

[80] Jiong Guo, Rolf Niedermeier, and Sebastian Wernicke. Fixed-parameter tractability results for full-degree spanning tree and its dual. In *Proceedings of the Second International Workshop on Parameterized and Exact Computation (IWPEC)*, volume 4169 of *LNCS*, pages 203–214. Springer, 2006.

[81] Jiong Guo, Rolf Niedermeier, and Sebastian Wernicke. Parameterized complexity of Vertex Cover variants. *Theory Comput. Syst.*, 41(3):501–520, 2007.

[82] Gregory Gutin, E. J. Kim, and Igor Razgon. Minimum leaf out-branching problems. *arxiv.org/abs/0801.1979*, 2008.

[83] Gregory Gutin, Ton Kloks, Chuan-Min Lee, and Anders Yeo. Kernels in planar digraphs. *J. Comput. Syst. Sci.*, 71(2):174–184, 2005.

[84] Sebastian Heinz. Sebastian heinz. complexity of integer quasiconvex polynomial optimization. *Journal of Complexity*, 21:543–556, 2005.

[85] Falk Hüffner, Christian Komusiewicz, Hannes Moser, and Rolf Niedermeier. Fixed-parameter algorithms for cluster vertex deletion. In *LATIN*, pages 711–722, 2008.

[86] H.A. Jung. On subgraphs without cycles in tournaments. *Combinatorial Theory and its Applications II*, pages 675–677, 1970.

[87] Iyad A. Kanj, Michael J. Pelsmajer, Ge Xia, and Marcus Schaefer. On the induced matching problem. In *Proceedings of the 25th Annual Symposium on Theoretical Aspects of Computer Science (STACS 2008)*, volume 08001, pages 397–408. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, Berlin, 2008.

[88] R. Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of Operations Research*, 12:415–440, 1987.

[89] K.B.Reid. On sets of arcs containing no cycles in tournaments. *Canad. Math Bulletin*, 12:261–264, 1969.

[90] K.D.Reid and E.T.Parker. Disproof of a conjecture of erdos and moser on tournaments. *J. Combin. Theory*, 9:225–238, 1970.

[91] Claire Kenyon-Mathieu and Warren Schudy. How to rank with few errors. In *STOC'07*, pages 95–103. ACM, 2007.

[92] L. Khachiyan and L. Porkolab. Integer optimization on convex semialgebraic sets. *Discrte Computational Geometry*, 23:207–224, 2000.

[93] Leonid Khachiyan. A polynomial algorithm in linear programming. *Soviet Math. Doklady*, 20(1):191–194, 1979.

[94] Joachim Kneis, Alexander Langer, and Peter Rossmanith. A new algorithm for finding trees with many leaves. In *ISAAC*, pages 270–281, 2008.

[95] Joachim Kneis, Daniel Mölle, Stefan Richter, and Peter Rossmanith. Divide-and-color. In *WG*, pages 58–67, 2006.

[96] Daniel Kobler and Udi Rotics. Polynomial algorithms for partitioning problems on graphs with fixed clique-width (extended abstract). In *SODA'01*, pages 468–476. ACM-SIAM, 2001.

[97] Daniel Kobler and Udi Rotics. Edge dominating set and colorings on graphs with fixed clique-width. *Discrete Appl. Math.*, 126(2-3):197–221, 2003.

[98] A. Koutsonas and D. M. Thilikos. Planar feedback vertex set and face cover: Combinatorial bounds and subexponential algorithms. In *34th Workshop on Graph-Theoretic Concepts in Computer Science (WG '08)*, volume 5344 of *Lecture Notes in Computer Science*, pages 264–274. Springer Verlag, 2008.

[99] Jan Kratochvíl and Mirko Kriválnek. On the computational complexity of codes in graphs. In *Proc. 13th MFCS*, volume 324 of *LNCS*, pages 396–404. Springer, 1988.

[100] Stefan Langerman and Pat Morin. Covering things with things. *Discrete & Computational Geometry*, 33(4):717–729, 2005.

[101] D. Lapoire. Recognizability equals definability, for every set of graphs of bounded tree-width. In *Proceedings 15th Annual Symposium on Theoretical Aspects of Computer Science*, pages 618–628. Springer Verlag, Lecture Notes in Computer Science, vol. 1373, 1998.

[102] H.W. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8:538–548, 1983.

[103] Daniel Lokshtanov. On the complexity of computing treelength. In *MFCS*, pages 276–287, 2007.

[104] Daniel Lokshtanov, Matthias Mnich, and Saket Saurabh. Linear kernel for planar connected dominating set. In *Proceedings of Theory and Applications of Models of Computation, (TAMC 2009)*. Springer, 2009.

[105] J.A. Makowsky, U. Rotics, I. Averbouch, and B. Godlin. Computing graph polynomials on graphs of bounded clique-width. In *WG'06*, LNCS, pages 191–204. Springer, 2006.

[106] Federico Mancini. Minimum fill-in and treewidth of split+ e and split+ v graphs. In *ISAAC*, pages 881–892, 2007.

[107] Dániel Marx. Chordal deletion is fixed-parameter tractable. In *WG*, pages 37–48, 2006.

[108] Kurt Mehlhorn. *Data Structures and Algorithms 2: Graph Algorithms and NP-Completeness*, volume 2 of *Monographs in Theoretical Computer Science. An EATCS Series*. Springer, 1984.

[109] W. Meyer. Equitable coloring. *American Mathematical Monthly*, 80:920–922, 1973.

[110] Bojan Mohar and Carsten Thomassen. *Graphs on Surfaces*. Johns Hopkins University Press, 2001.

[111] Hannes Moser, Venkatesh Raman, and Somnath Sikdar. The parameterized complexity of the unique coverage problem. In *Proc. 18th ISAAC*, volume 4835 of *LNCS*, pages 621–631. Springer, 2007.

[112] Hannes Moser and Somnath Sikdar. The parameterized complexity of the induced matching problem in planar graphs. In *Proceedings First Annual International WorkshopFrontiers in Algorithmics (FAW)*, volume 4613 of *LNCS*, pages 325–336. Springer, 2007.

[113] G. J. Woeginger N. Alon, Y. Azar and T. Yadid. Approximation schemes for scheduling on parallel machines. *J. of Scheduling*, 1:55–66, 1998.

[114] M. Naor, L. J. Schulman, and A. Srinivasan. Splitters and near-optimal derandomization. In *FOCS*, pages 182–191, 1995.

[115] G. L. Nemhauser and L. E. Trotter. Vertex packing: structural properties and algorithms. *Math. Programming*, 8:232–248, 1975.

[116] Rolf Niedermeier. *An Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.

[117] A. Nilli. Perfect hashing and probability. *Combinatorics, Probability and Computing*, 3:407–409, 1994.

[118] Christos H. Papadimitriou and Mihalis Yannakakis. On limited nondeterminism and the complexity of the v.c dimension (extended abstract). In *Structure in Complexity Theory Conference*, pages 12–18, 1993.

[119] V. Raman and S. Saurabh. Parameterized algorithms for feedback set problems and their duals in tournaments. *Theoretical Computer Science*, 351(3):446–458, 2006.

[120] B. Reed, K. Smith, and A. Vetta. Finding odd cycle transversals. *Operations Research Letters*, 32:229–301, 2004.

[121] Neil Robertson and Paul D. Seymour. Graph minors .xiii. the disjoint paths problem. *J. Comb. Theory, Ser. B*, 63(1):65–110, 1995.

[122] Neil Robertson and Paul D. Seymour. Graph minors. xx. wagner's conjecture. *J. Comb. Theory, Ser. B*, 92(2):325–357, 2004.

[123] Neil Robertson, Paul D. Seymour, and Robin Thomas. Quickly excluding a planar graph. *J. Comb. Theory, Ser. B*, 62(2):323–348, 1994.

[124] S. Seshu and M.B. Reed. *Linear Graphs and Electrical Networks.* Addison-Wesley, 1961.

[125] Paul D. Seymour and Robin Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2):217–241, 1994.

[126] M. Sipser. *Introduction to the Theory of Computation.* International Thomson Publishing, 1996.

[127] P. Slater. Inconsistencies in a schedule of paired comparisons. *Biometrika*, 48:303–312, 1961.

[128] Christian Sloper and Jan Arne Telle. An overview of techniques for designing parameterized algorithms. *Comput. J.*, 51(1):122–136, 2008.

[129] J. Spencer. Optimal ranking of tournaments. *Networks*, 1:135–138, 1971.

[130] J. Spencer. Optimal ranking of unrankable tournaments. *Period. Math. Hungar.*, 11(2):131–144, 1980.

[131] Larry J. Stockmeyer. The polynomial-time hierarchy. *Theor. Comput. Sci.*, 3(1):1–22, 1976.

[132] Stéphan Thomassé. A quadratic kernel for feedback vertex set. In *Proc. 20th SODA*, pages 115–119. ACM/SIAM, 2009.

[133] A. van Zuylen. Deterministic approximation algorithms for ranking and clusterings. Technical Report 1431, Cornell ORIE, 2005.

[134] A. van Zuylen, Rajneesh Hegde, Kamal Jain, and David P. Williamson. Deterministic pivoting algorithms for constrained ranking and clustering problems. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 405–414, 2007.

[135] K. Wagner. Uber einer eigenschaft der ebener complexe. *Math. Ann.*, 14:570–590, 1937.

[136] Ryan Williams. Finding paths of length k in $O^*(2^k)$ time. *CoRR*, abs/0807.3026, 2008.

[137] D.H. Younger. Minimum feedback arc sets for a directed graph. *IEEE Trans. Circuit Theory*, 10:238–245, 1963.