SYSTEMATIC KERNELIZATION IN FPT ALGORITHM DESIGN

Elena Prieto Rodríguez

Lic. Ciencias Matemáticas

*A thesis submitted in partial fulfilment
of the requirements for the degree of*

Doctor of Philosophy

School of Electrical Engineering
and Computer Science

The University of Newcastle
Callaghan N.S.W. 2308
Australia

March, 2005

The UNIVERSITY
of NEWCASTLE
AUSTRALIA

I LOOK AHEAD

I hereby certify that the work embodied in this thesis is the result of original research and has not been submitted for a higher degree to any other University or Institution.

Elena Prieto Rodríguez

ACKNOWLEDGEMENTS

I would like to thank Mike Fellows for stimulating my interest in research and guiding me through the exciting path of parameterized complexity. I am very fortunate to have been given the opportunity of working with him. I would also like to give special thanks to Christian Sloper for the very interesting and insightful conversations about almost all topics discussed in this thesis.

During the past three years I have collaborated with a number of people I would like to extend my gratitude to. From all of them I have learnt a great deal about computer science, mathematics and the art and craft of writing papers. In particular I would like to thank Fran Rosamond for carefully reading various drafts of this dissertation, Henning Fernau for pointing out several extremal combinatorics results I have used, Luke Mathieson for being so patient with my English grammar.

I would also like to mention Ulrike Stege, Jan Arne Telle and Rolf Niedermeier who kindly invited me to visit their respective universities. They all proved to me how wonderful scientific collaboration can be.

In the University of Newcastle, I will be forever grateful to Pablo Moscato for thoroughly revising the last drafts and making sure I finished on time.

In a personal level I would like to mention the support received from all my friends in Newcastle (go Black Stellas!), my extended family in Spain, my friend Carolina and especially my husband Rhyall. Thank you for your unconditional support through thick and thin. Thank you for being there every single time. I wouldn't have finished this thesis without your love, encouragement and patience.

Muchas gracias mamá, papá y Manolo. Aunque el pavo todavía me persigue ('quítamelo, quítamelo'), sé que no habría llegado aquí sin vuestro cariño. Os lo debo todo.

To Rhyall.

CONTENTS

ABSTRACT

Data reduction is a preprocessing technique which makes huge and seemingly intractable instances of a problem small and tractable. This technique is often acknowledged as one of the most powerful methods to cope with the intractability of certain NP-complete problems. Heuristics for reducing data can often be seen as reduction rules, if considered from a parameterized complexity viewpoint. Using reduction rules to transform the instances of a parameterized problem into equivalent instances, with size bounded by a function of the parameter, is known as kernelization.

This thesis introduces and develops an approach to designing FPT algorithms based on effective kernelizations. This method is called the *method of extremal structure*. The method operates following a paradigm presented by two lemmas, the kernelization and boundary lemmas. The boundary lemma is used to find theoretical bounds on the maximum size of an instance reduced under an adequate set of reduction rules. The kernelization lemma is invoked to decide the instances which are larger than $f(k)$ for some function f depending only on the parameter k.

The first aim of the *method of extremal structure* is to provide a systematic way to discover reduction rules for fixed-parameter tractable problems. The second is to devise an analytical way to find theoretical bounds for the size of kernels for those problems. These two aims are achieved with the aid of combinatorial extremal arguments.

Furthermore, this thesis shows how the *method of extremal structure* can be applied to effectively solve several NP-complete problems, namely MAX CUT, MAX LEAF SPANNING TREE, NONBLOCKER, s-STAR PACKING, EDGE-DISJOINT TRIANGLE PACKING, MAX INTERNAL SPANNING TREE and MINIMUM MAXIMAL MATCHING.

LIST OF FIGURES

LIST OF TABLES

1

INTRODUCTION

1.1 MOTIVATION

The search for better and faster algorithms for optimization problems has been the focus of the efforts of countless mathematicians and computer scientists for the past hundred years. Since the beginning of this search it was noticed that some problems seemed intrinsically harder to solve than others. The need to classify these problems by their apparent difficulty led to the creation of the field of computational complexity theory. In the context of this theory, problems which can be solved by polynomial-time deterministic algorithms are said to belong to the class P and those which can be solved nondeterministically in polynomial time are said to belong to the class NP. Within NP the class of NP-complete problems encompasses all those problems that are at least as hard to solve as any other problem in the class. It is generally believed that these NP-complete problems are not solvable in deterministic polynomial time [G02].

Many of these NP-complete problems arise naturally in fields of application such as networking, biology, systems design, operations research, etc., and thus it is of extreme importance to solve them, and solve them using the best possible algorithms. Towards this end, many different approaches have proved useful. One of these approaches is to explore various parameterizations, determine whether the resulting parameterized problems are fixed-parameter tractable, and if so, attempt to find the best possible FPT algorithms to solve them [DF99, Nie02]. Another approach is to reduce the size of the input data. Weihe [Wei98] presented an example that deals with an NP-complete problem arising in

the context of the European railroad network. This network has 25,000 stations, 140,000 trains and 1,600,000 train stops. In a preprocessing phase, two data reduction rules were exhaustively applied. When these rules could no longer be applied the network was split into connected components of size seven or less. For each of these components a brute-force algorithm sufficed to solve the problem optimally.

Efficient preprocessing and parameterized complexity converge in the FPT algorithm design area of kernelization. Heuristics for reducing data can often be seen as reduction rules if considered from a parameterized complexity viewpoint. Using reduction rules to transform the instances of a parameterized problem into equivalent instances with size bounded by a function of the parameter is known as kernelization.

This thesis exposits a method to find fixed-parameter tractable algorithms in a systematic way based on effective kernelizations. This method is called the *method of extremal structure*. We study this method and several algorithmic variants, demonstrating how it can be used to address several NP-complete problems.

The ideas which constitute the core of the *method of extremal structure* first appeared in 2000 in a paper by Fellows, McCartin, Rosamond, and Stege, *'Coordinatized Kernels and Catalytic Reductions: An Improved* FPT *Algorithm for Max Leaf Spanning Tree and Other Problems'* [FM+00]. In this paper, the authors introduced a new method that they called the *method of coordinatized kernels*. This method was based on kernelization techniques and ideas from extremal combinatorics. With the *method of coordinatized kernels* a model was proposed which could be used to solve a wide variety of problems. This model seeks to transform, in polynomial time, instances (I, k) of a problem $\mathcal{P}$ into equivalent smaller instances (I', k'). These transformations are known as reduction rules. Once the instances are no longer susceptible to these transformations (the instances are *reduced*), a *kernelization lemma* is invoked to decide the instances which are larger than $f(k)$ for some function f depending only on the parameter k.

The first aim of the *method of extremal structure* is to provide a systematic way to guide

the process of discovering reduction rules for fixed-parameter tractable problems. The second is to find an easy way to obtain theoretical bounds for the size of kernels for those problems. These aims are achieved with the aid of combinatorial extremal arguments via a *boundary lemma*.

1.2   DEFINITIONS

1.2.1   GRAPH THEORY

The notation we use coincides with that of Bollobás [Bol78], [Bol98] and Diestel [Die97].

A graph is denoted $G = (V, E)$. When we refer to the set of vertices of the graph we will write $V(G)$ or simply V if it is clear from the context what graph we are referring to. An *edge* between two vertices u and v is denoted (u, v). The set $E(G)$ denotes the set of edges of the graph, although it will generally be written as E. We will assume all graphs are simple (at most one edge exists between two vertices) and loopless (every edge must have two different endpoints). For any two subsets of vertices $X \subseteq V$ and $Y \subseteq V$, the set of all edges in E with one endpoint in X and the other in Y will be written $E(X, Y)$.

A graph $G' = (V', E')$ is a *subgraph* of a graph $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. A subgraph *induced* by a subset of vertices $S \subseteq V$ will be denoted by $\langle S \rangle$ and consists of the vertices in S plus all the edges in E which have both endpoints in S. The edges of that graph will be noted $E(S)$. The *size* of a set S will be denoted $|S|$.

Let $G = (V, E)$ and $G' = (V', E')$ be two graphs. We say that G and G' are *isomorphic* if there exists a bijection $\varphi : V \to V'$ such that $(u, v) \in E$ if and only if $(\varphi(u), \varphi(v)) \in E'$ for all u and v in V.

Two vertices u and v in G are *adjacent*, or *neighbors*, if $(u, v) \in E$. $N(v)$ denotes the *open neighborhood* of a vertex v, i.e., $N(v) = \{u | (u, v) \in E\}$. Then, $N[v] := N(v) \cup \{v\}$ is the *closed neighborhood* of v. Given a subset of vertices S, we also use $N(S)$ and $N[S]$ to denote the union of open and closed neighborhoods of vertices in S, respectively.

The degree of a vertex v is the number of vertices in its open neighborhood, that is $deg(v) = |N(v)|$. If $deg(v) = 0$ then v is termed *isolated*. Given a subset of vertices $S \subseteq V(G)$, we define $deg_{|S}(v) = |N(v) \cap S|$.

We say that $u \in V$ is a *common neighbor* of a pair of distinct vertices $\{w_1, w_2\}$ if both (u, w_1) and (u, w_2) are in $E(G)$. In this situation, if there exists an edge between w_1 and w_2, then we say that u *spans* the edge (w_1, w_2).

If $G = (V, E)$ is a graph and $v \in V$, then $G \smallsetminus v$ denotes the graph obtained from G by deleting v (and of course all incident edges). Conversely, $G \cup v$ is obtained from G by adding a (new) isolated vertex v; and, given two vertices $v \neq v'$ from G, we arrive at $G \cup (v, v')$ by adding the edge (v, v') to E. Similarly, $G \smallsetminus H$ for $H \subseteq V$ denotes the graph obtained from G by deleting all the vertices in H together with all incident edges to H. If $v \neq v'$ are two vertices in $G = (V, E)$, then $G_{[v \leftrightarrow v']}$ denotes the graph obtained from G by merging v and $v' \in V$ into a new vertex $v \leftrightarrow v'$ with

$$N(v \leftrightarrow v') = (N(v) \cup N(v')) \smallsetminus \{v, v'\}.$$

A *path* $P = (V', E')$ is a non-empty graph of the form $V' = \{x_0, x_1, \ldots, x_k\}$, $E' = \{(x_0, x_1), (x_1, x_2), \ldots, (x_{k-1}, x_k)\}$, where the x_i are all distinct. The vertices x_0 and x_p are called the endpoints of P. The length of P is defined as its number of edges, and the path of length k is denoted by P_k.

If all vertices in a graph $G = (V, E)$ are pairwise adjacent, then G is *complete*. A complete graph on n vertices is denoted K_n. A K_3 is a triangle. A graph $G = (V, E)$ is *bipartite* if it admits a partition of V into two subsets such that no two vertices within the same subset are adjacent. A bipartite graph such that every pair of graph vertices in the two sets are adjacent is called a *complete bipartite* graph. If one of the sets has cardinality one then the complete bipartite graph is called a *star* and denoted $K_{1,n}$.

A graph is *planar* if it can be drawn in a plane without graph edges crossing. A graph

is *regular* if all its vertices have the same degree. A non-empty graph G is *connected* if any two of its vertices are linked by a path in G. A maximal connected subgraph of G is called a *component* of G.

A *bridge* $(u, v) \in E(G)$ in a connected component of a graph G is an edge whose deletion disconnects the component. An *articulation vertex* of a connected graph is a vertex whose removal will disconnect the graph. In general, an articulation vertex is a node of a graph whose removal increases the number of components. Note that since at least a new component is generated by the removal of u, we can always group the newly generated components into two disjoint components X and Y, not necessarily connected themselves. In this context we say that X and Y are the two components *arising* from the removal of u in G. A connected graph with no articulation vertices is said to be *biconnected*. If $A, B \subseteq V$ and $X \subseteq V \cup E$ are such every path from A to B in G contains a vertex or an edge from X, we say that X *separates* the sets A and B in G. In general, such a set X will be called a *separator*.

If $P = x_0, x_1, \ldots, x_k$ is a path and $k \geq 2$, then the graph $P \cup (x_k, x_0)$ is called a *cycle*. A graph not containing any cycles is called *acyclic*. A *tree* is a connected acyclic graph. A *spanning tree* T of a graph $G = (V, E)$ is a subgraph of G that contains all the vertices of G and is a tree.

We say that a vertex $v \in V(T)$ is *internal* in T if there exist at least two vertices u and w such that $u, w \in V(T)$ and $(u, v), (v, w) \in E(T)$. Any vertex in T which is not internal in T is a *leaf*.

Each edge (u, v) in a spanning tree T of a graph G induces an orientation in T. This orientation induces two subtrees in T, T_u and T_v, which we use to denote the subtree to the left and the subtree to the right of the edge (u, v). Two vertices in G are said to be in different subtrees of (u, v) if one of them is in the left subtree and the other is in the right subtree.

A *matching* on a graph G is a set of edges in $E(G)$ such that no two of them share a common endpoint. A *perfect matching* is a matching such that all vertices in the graph are endpoints of some edge in the matching.

We use the following lemma (based on Stirling's formula), once we have a linear kernel for a problem, to calculate the number of subsets of size k contained in the kernel. It is quite handy and often gives a considerably smaller bound than the "naive" $\mathcal{O}(2^{ak})$:

Lemma 1.1 *For any fixed $a > 1$,*

$$\binom{ak}{k} \approx a^k \left(\frac{a}{a-1} \right)^{(a-1)k} = \left(\frac{a^a}{(a-1)^{a-1}} \right)^k$$

Proof of Lemma 1.1.

$$
\begin{aligned}
\binom{ak}{k} &= \frac{(ak)!}{(ak-k)!\,k!} \\[2mm]
&\approx \frac{\sqrt{2\pi ak}}{\sqrt{2\pi k}\,\sqrt{2\pi(a-1)k}} \cdot \left(\frac{ak}{e} \right)^{ak} \cdot \left(\frac{e}{k} \right)^k \cdot \left(\frac{e}{(a-1)k} \right)^{(a-1)k} \\[2mm]
&= \frac{\sqrt{a}}{\sqrt{2\pi(a-1)k}}\, a^k \left(\frac{a}{a-1} \right)^{(a-1)k} \\[2mm]
&\leq a^k \left(\frac{a}{a-1} \right)^{(a-1)k} \\[2mm]
&= \left(\frac{a^a}{(a-1)^{a-1}} \right)^k
\end{aligned}
$$

$\square$

1.2.2 COMPLEXITY THEORY

The notation we use coincides with that of Garey and Johnson [GJ79] and Ausiello et al. [AC+99].

A *problem* $\mathcal{P}$ is a relation between a set $\mathcal{I}$ of problem instances and SOL, a set of problem solutions.

There are different kinds of problems depending on the characteristics of the sets $\mathcal{I}$ and SOL and the relation $\mathcal{P}$. In this thesis we will be working with a subclass of problems called decision problems. A problem $\mathcal{P}$ is a *decision problem* if $SOL =\{$YES,NO$\}$. Hence, a problem $\mathcal{P}$ is a decision problem if the set $\mathcal{I}$ can be partitioned into a set $Y_{\mathcal{P}}$ of YES-instances and a set $N_{\mathcal{P}}$ of NO-instances, and the problem asks, for any instance $I \in \mathcal{I}$, to verify whether $I \in Y_{\mathcal{P}}$.

Definition 1.1 *We say that a decision problem is in* P *if it can be solved in polynomial time by a deterministic Turing Machine. We say that a problem is in* NP *if it can be solved in polynomial time by a nondeterministic Turing Machine.*

Definition 1.2 *We say that a decision problem $\mathcal{P}$ is reducible to another problem $\mathcal{P}'$, if there exists a polynomial-time deterministic algorithm which transforms instances $I' \in \mathcal{I}'$ of $\mathcal{P}'$ into instances $I \in \mathcal{I}$ of $\mathcal{P}$, such that $I \in Y_{\mathcal{P}}$ if and only if $I' \in Y_{\mathcal{P}'}$.*

Definition 1.3 *A decision problem $\mathcal{P}$ is NP-complete if*

1. *$\mathcal{P}$ is in NP and*

2. *$\mathcal{P}$ is NP-hard, i.e. every other problem in NP is reducible to it.*

A consequence of this definition is that if we had a polynomial time algorithm for $\mathcal{P}$, we could solve all problems in NP in polynomial time.

In this thesis we address a number of problems and compare our results with previous work. When doing so we often resort to known approximation results for those problems. Approximation results deal with another subclass of problems, optimization problems. An

optimization problem $\mathcal{P}$ is characterized by a quadruple of objects $(\mathcal{I}, SOL, m_{\mathcal{P}}, goal_{\mathcal{P}})$, where:

1. the set $\mathcal{I}$ is the set of instances of $\mathcal{P}$;

2. the function SOL is a function that associates to any instance $I \in \mathcal{I}$ the set of feasible solutions of I;

3. the function $m_{\mathcal{P}}$ is a measure function, defined for pairs (I, S) such that $I \in \mathcal{I}$ and $S \in SOL(I)$. For every such pair (I, S), $m_{\mathcal{P}}(I, S)$ provides a positive integer which is the value of the feasible solution S;

4. $goal_{\mathcal{P}} \in \{\text{MIN}, \text{MAX}\}$ specifies whether $\mathcal{P}$ is a minimization or a maximization problem.

Given an instance I, we denote by $SOL^*(I)$ the set of optimal solutions of I. That is, for every $S^*(I)$ such that $S^*(I) \in SOL^*(I)$:

$$m_{\mathcal{P}}(I, S^*(I)) = goal_{\mathcal{P}}\{v | v = m_{\mathcal{P}}(I, S) \land S \in SOL(I)\}$$

The value of any optimal solution $S^*(I)$ of I will be denoted as $m_{\mathcal{P}}^*(I)$. It is important to notice that any optimization problem $\mathcal{P}$ has an associated decision problem $\mathcal{P}_D$ [AC+99].

Definition 1.4 *An* optimization *problem* $\mathcal{P} = (\mathcal{I}, SOL, m_{\mathcal{P}}, goal_{\mathcal{P}})$ *belongs to the class* NPO *if the following holds:*

1. *the set of instances $\mathcal{I}$ is recognizable in polynomial time;*

2. *there exists a polynomial q such that, given an instance $I \in \mathcal{I}$, for any $S \in SOL(I), |S| \leq q(|I|)$. Moreover, for any I and for any S with $|S| \leq q(|I|)$, it is decidable in polynomial time whether $S \in SOL(I)$;*

3. *the function $m_{\mathcal{P}}$ is computable in polynomial time.*

Theorem 1.1 *[AC+99] For any optimization problem $\mathcal{P}$ in* NPO*, the corresponding decision problem $\mathcal{P}_D$ belongs to* NP*.*

Definition 1.5 *An approximation algorithm* **A** *for an* NPO *problem $\mathcal{P}$ is an algorithm which given any instance I returns a feasible solution* **A**$(I) \in SOL(I)$.

Definition 1.6 *Given an* NPO *problem $\mathcal{P}$, an instance of the problem I and a feasible solution S, the performance ratio of S with respect to I is defined as*

$$R(I, S) = \max\left(\frac{m_{\mathcal{P}}(I, S)}{m_{\mathcal{P}}^*(I)}, \frac{m_{\mathcal{P}}^*(I)}{m_{\mathcal{P}}(I, S)}\right).$$

Definition 1.7 *An approximation algorithm* **A** *for an* NPO *problem $\mathcal{P}$ is an ε-approximate algorithm if given any instance I, the performance ratio of the approximate solution* **A**(I) *is bounded by ε, that is:*

$$R(I, \mathbf{A}(I)) \leq \varepsilon.$$

Definition 1.8 *The class* APX *is the class of* NPO *problems $\mathcal{P}$ such that, for some $\varepsilon \geq 1$, there exists a polynomial time ε-approximate algorithm for $\mathcal{P}$.*

A problem is APX-*hard* if there exists some constant $\varepsilon > 0$ such that it is NP-hard to approximate the problem within a factor of $(1 + \varepsilon)$. A problem which is both in APX and APX-hard is called an APX-*complete problem.*

Definition 1.9 *An approximation algorithm* **A** *for an* NPO *problem $\mathcal{P}$ is said to be a polynomial-time approximation scheme for $\mathcal{P}$ if for any instance I and rational value $\varepsilon > 1$, when* **A** *is applied to (I, ε) it returns an ε-approximate solution of I in time polynomial in $|I|$.*

Definition 1.10 *The class* PTAS *is the class of* NPO *problems that admit a polynomial-time approximation scheme.*

The class MAX SNP (containing e.g. MAXIMUM 3-SATISFIABILITY and MAXIMUM CUT) has been shown to be identical to APX [KM+99]. Therefore, we use the term APX-complete instead of MAX SNP-complete and APX-hard instead of MAX SNP-hard.

1.3   PARAMETERIZED COMPLEXITY

In classical computational complexity theory, a decision problem is specified by two items: the input to the problem and the question to be answered. Decision problems such as VERTEX COVER and DOMINATING SET are thus generally expressed as follows:

VERTEX COVER [1]

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

Question: Is there a subset $V' \subset V$ of size k such that for all edges

$(v_1, v_2) \in E$ either $v_1 \in V'$ or $v_2 \in V'$?

DOMINATING SET [2]

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

Question: Is there a subset $V' \subset V$ of size k such that for each vertex

$u \in V$ there is a $v \in V'$ such that the edge $(u, v) \in E$?

Although both problems are NP-complete from a classical complexity viewpoint, the introduction of k as a parameter provides a new approach to their complexity. After many rounds of improvement on algorithms which efficiently solve the problems, the best known algorithm for VERTEX COVER runs in time $O(1.2745^k k^4 + k|V|)$ [CG04]. Fomin et al. [FKW04] give the best exact algorithm to date, which has a time complexity of $\mathcal{O}(1.93782^{|V|})$, for DOMINATING SET.

---

[1]The set V' is called a *vertex cover* for G.
[2]The set V' is called a *dominating set* for G.

In parameterized complexity, a third element of information is added: the aspects of the input which constitute the parameter. In this context, Downey and Fellows introduce the following definition:

Definition 1.11 *A parameterized problem is a subset* $\mathcal{P} \subseteq \Sigma^* \times \Sigma^*$

For notational convenience we will consider that $\mathcal{P} \subseteq \Sigma^* \times \mathbb{N}$, since the parameter is a positive integer in all the examples shown in this thesis.

Definition 1.12 *(Fixed Parameter Tractability) A parameterized problem $\mathcal{P}$ is* fixed-parameter tractable *if there is an algorithm that correctly decides for input $(I, k) \in \Sigma^* \times \mathbb{N}$, whether $(I, k) \in \mathcal{P}$ in time $f(k) + n^\alpha$, where n is the size of the input I, $|I| = n$, k is the parameter, α is a constant (independent of k and n) and f is an arbitrary function.*

The class of fixed-parameter tractable parameterized problems is denoted FPT. It is to be noted that FPT is unchanged if the definition above is modified by replacing $f(k) + n^\alpha$ by $f(k)n^\alpha$ [CC+97].

An algorithm running in time $f(k) + n^\alpha$ to decide a problem $\mathcal{P}$ is called an FPT *algorithm*. If the FPT algorithm outputs YES then the instance $(I, k) \in \mathcal{P}$ and we say that (I, k) is a YES-instance for $\mathcal{P}$. Otherwise we say (I, k) is a NO-instance.

Using Definition 1.12 and the algorithm in [CG04] we can see that VERTEX COVER is fixed-parameter tractable. However, it is not believed that all NP-complete problems are fixed-parameter tractable. In fact, DOMINATING SET is not believed to be likely to be in FPT.

Apparent fixed-parameter *intractability* is established via a completeness program. The main sequence of parameterized complexity classes is

$$FPT \subseteq W[1] \subseteq W[2] \subseteq \ldots \subseteq W[t] \subseteq \ldots \subseteq W[P] \subseteq AW[P] \subseteq XP$$

This sequence is commonly termed the W-hierarchy . The complexity class $W[1]$ is the parameterized analog of NP. The defining complete problem for $W[1]$ is:

k-STEP NDTM HALTING PROBLEM

Instance: A nondeterministic Turing machine (NDTM) M and a string x

Parameter: A positive integer k

Question: Does M have a computation path accepting x in at most k steps?

In the same sense that NP-completeness of POLYNOMIAL TIME NDTM HALTING PROB-LEM provides us with strong evidence that no NP-complete problem is likely to be solvable in polynomial time, $W[1]$-completeness of k-STEP NDTM HALTING PROBLEM provides us with strong evidence that no $W[1]$-complete problem is likely to be fixed-parameter tractable [DF95b].

Heavy computational machinery is necessary to prove that DOMINATING SET is complete for the class $W[2]$ and thus not likely to be in FPT. For a more in-depth study of the $W[t]$ hierarchy and its structural complexity we refer readers to the comprehensive study on parameterized complexity by Downey and Fellows [DF99].

Definition 1.13 *A parameterized problem $\mathcal{P}$ is many:one parametrically reducible to a parameterized problem $\mathcal{P}'$ if there is an FPT algorithm that transforms (I, k) into (I', k') so that:*

1. *$(I, k) \in \mathcal{P}$ if and only if $(I', k') \in \mathcal{P}'$, and*

2. *$k' \leq g(k)$ (where g is an unrestricted function; k' is purely a function of k)*

Since the running time of a fixed-parameter tractable algorithm is a function of both $n = |I|$ and k, it will thus be expressed as $\mathcal{O}(f(k)n^{\alpha})$ where k is the fixed parameter. There is an alternative notation used by Woeginger [Woe03] for expressing the running time of

algorithms. In this notation the running time is expressed as $\mathcal{O}^*(f(k)) = \mathcal{O}(f(k)n^\alpha)$. Throughout this thesis we will also use this modified big-Oh notation that suppresses all other (polynomially bounded) terms that depend on the size of the instance I.

1.4 KERNELIZATION

Many different techniques are used to develop FPT algorithms [DF99]. Informally, we can think of these techniques as being divided into two groups: those contributing to the $f(k)$ *game*; and those contributing to the *kernelization game* [Fel03].

The techniques involved in the $f(k)$ *game* aim to improve the function $f(k)$ defined in the previous section. The usefulness of this game is emphasized using the $\mathcal{O}^*(f(k))$ notation when giving the running time of the algorithms. Improvements to the algorithms in the $f(k)$ game may arise from the use of methods for integer linear programming, shrinking search trees by dynamic programming, color coding, dynamic programming on tree decompositions, etc. For a review of these techniques we refer the reader to [Nie02].

In this thesis we will focus on the other game, the *kernelization game*. A kernelizable problem is formally defined as follows:

Definition 1.14 *A parameterized problem $\mathcal{P}$ is kernelizable if there is a parametric transformation of $\mathcal{P}$ to itself that satisfies:*

1. *The running time of the transformation of (I, k) into (I', k'), where $|I| = n$, is bounded by a polynomial $q(n, k)$ (so that in fact this is a polynomial time transformation of $\mathcal{P}$ to itself, considered classically, although with the additional structure of a parametric reduction),*

2. $k' \leq k$, *and*

3. $|I'| \leq h(k)$, *where h is an arbitrary function.*

The *kernel* of a problem is the transformed instance (I', k') defined above.

It is important to note the following result, which gives us the link between the two games:

Lemma 1.2 *[DFS99] A recursive parameterized problem $\mathcal{P}$ is in* FPT *if and only if it is kernelizable.*

Proof of Lemma 1.2.

$(\Leftarrow:)$ If the parameterized problem $\mathcal{P}$ is kernelizable then the size of any transformed instance can be bounded by a function of k. Hence the running time of an algorithm deciding a transformed instance (I', k') would be dependent only on k. Since the transformation can be performed in polynomial time, the total running time of the algorithm deciding an instance (I, k) would be $\mathcal{O}(n^\alpha + f(k))$ and thus $\mathcal{P}$ is in FPT.

$(\Rightarrow:)$ A problem is in FPT, if there is a constant $\alpha > 1$ and a function $f(k)$ such that we can determine if $(I, k) \in \mathcal{P}$ in time $\mathcal{O}(n^\alpha + f(k))$. Our kernelization algorithm for $\mathcal{P}$ considers two cases:

1. If $k < f^{-1}(n)$ then in time $\mathcal{O}(n^\alpha)$ we can simply answer the question.

2. If $k \geq f^{-1}(n)$ then $n \leq f(k)$ and the instance (I, k) already belongs to the problem kernel.

$\square$

All techniques are worth using and combining, as they may lead to substantial improvements in the algorithms we design and also to new strategies for practical implementations of these algorithms [AL+03, DR+03].

As pointed out by Fellows [Fel03], both games can also be considered as giving a solid mathematical foundation to two basic techniques for designing good practical heuristics which solve "hard problems":

- "Branch and cut" and similar paradigms often correspond to search tree based parameterized algorithms, the parameter being the height of the search tree. Heuristics for cutting off subtrees can often be analyzed in the parameterized setting.

- "Data reduction" is the principal preprocessing technique which often makes seemingly intractable huge instances small and tractable. Heuristics for reducing data can be often seen as "reduction rules", if considered from a parameterized viewpoint.

The idea of kernelization can be illustrated using the VERTEX COVER problem. Samuel Buss' preprocessing algorithm [DF99] in Table 1.1 can be applied to any instance (G, k) of VERTEX COVER.

```
Step 1.  Find all vertices in G of degree greater than
         k, let p be the number of such vertices.
Step 2.  If p > k, then answer NO and halt.
Step 3.  Else, discard all p vertices of degree at least
         k and edges incident to these vertices and let
         k' = k - p.
Step 4.  If the resulting graph G' has more than k'(k + 1)
         vertices, or more than k'k edges, answer NO and
         halt.
```

Table 1.1: Preprocessing algorithm for VERTEX COVER.

When the algorithm terminates G' has size bounded by $h(k) = k'(k+1)$. Hence, VERTEX COVER is kernelizable. Steps 1 and 2 of the algorithm are correct. Any vertex with degree higher than k is part of a k-vertex cover, as otherwise all its neighbors must be included in the vertex cover. If we have more than k of those, we immediately have a NO-instance. Step 3 of the algorithm in Table 1.1 is also correct. If v has degree greater than k, it would always be better to include v in V' instead of all its neighbors. Further, k' vertices of degree at most k can cover at most $k'k$ edges, incident to $k'(k + 1)$ vertices. This last fact provides the $k'(k + 1) = (k - p)(k + 1) = \mathcal{O}(k^2)$ bound claimed in Step 4. This type

of kernel is known as *quadratic* as it is quadratic in k. A better kernelization for VERTEX COVER is given by a result of Nemhauser and Trotter [NT75] providing a size $2k$ kernel, linear in k [CKJ01].

Deleting vertices of high degree in VERTEX COVER is what we call a *reduction rule*. Formally, a reduction rule is a parametric transformation of an instance (I, k) into (I', k'), where $k' = f(k)$ for some function f. If $|I| = n$, the running time of the transformation must be bounded by a polynomial $q(n, k)$, so that, in fact, this is a polynomial time transformation as stated in Definition 1.14.

When we prove the *soundness* of a reduction rule for a problem $\mathcal{P}$, what we prove is that (I, k) is a YES-instance for $\mathcal{P}$ if and only if (I', k') is a YES-instance for $\mathcal{P}$.

Continuing with the VERTEX COVER example, if the instance is (G, k) and G has a *pendant vertex* v adjacent to a vertex u (i.e., v has degree one and u is its neighbor in G), then it would not be worse, and could be better, to include u in the vertex cover instead of v. Hence, (G, k) can be reduced to $(G', k - 1)$, where $G' = G \smallsetminus u$ is obtained from G by deleting u.

One of the most important features of reduction rules is that they can *cascade*: after we apply one of them another one may be triggered. An *irreducible* instance (also called *reduced* instance) of a problem $\mathcal{P}$ with respect to a set of reduction rules R is an instance which is no longer susceptible to the application of any of the reduction rules in R. However, we must ensure that the different reduction rules do not trigger each other ending up in circular arguments. In the case of the VERTEX COVER problem, some more complicated and much less obvious reduction rules can be found in the current state-of-the-art FPT algorithms (see [BFR98, DFS99, NR99, Ste00, CKJ01, CG04]).

1.5 CROWN REDUCTIONS

There is a type of reduction rule which plays a significant role in this thesis, the *crown reduction*. Traditionally, most reduction rules were limited to reduction rules based on the structure of a fixed number of vertices. In [CFJ04] another type of reduction rule was introduced which uses global structures of the following type:

Definition 1.15 *A crown decomposition of a graph $G = (V, E)$ is a partition of the set of vertices of the graph into three non-empty sets H, C and X with the following properties:*

1. H (the head) *is a separator in G such that there are no edges in G between vertices belonging to C and vertices belonging to X.*

2. $C = C_u \cup C_m$ (the crown) *is an independent set in G, and C_u, C_m form a partition of C.*

3. $|C_m| = |H|$, *and there is a perfect matching between C_m and H.*

Some problems whose instances admit crown decompositions can have reduction rules using these decompositions. These reduction rules are called *crown reductions*.

We now illustrate, using the VERTEX COVER example, how an instance (G, k) of this problem that admits a crown decomposition, can be reduced to an equivalent instance (G', k') where G' is obtained from G by deleting both sets C and H and $k' = k - |H|$.

Reduction Rule 1 *[CFJ04] Let (G, k) be a graph instance that admits a crown decomposition (C, H, X). Let G' be G after deleting H and C. Transform (G, k) into $(G', k - |H|)$.*

Soundness Lemma 1 Reduction Rule 1 is sound.

   Proof of Soundness Lemma 1.

($\Leftarrow$:) If G' has a vertex cover of size $k - |H|$ then G has a vertex cover of size k as we can place the vertices in H in the vertex cover. Since C is an independent set, the vertices in H will cover all edges we are adding to the graph.

($\Rightarrow$:) We must show that if G has a vertex cover of size k then G' has a vertex cover of size $k - |H|$. Since H is matched into C there exist $|H|$ independent edges in $\langle C \cup H \rangle$. All these edges must be covered by some vertex and, as they are independent, these vertices are different. Thus, when $C \cup H$ is deleted, the size of k decreases by at least $|H|$. Therefore, $(G', k - |H|)$ is a YES-instance for VERTEX COVER. □

The *crown reduction* generalizes the reduction rule for a degree one vertex for VERTEX COVER, to one that admits a global structure [Fel03]. A crown is said to be *nontrivial* if both H and C are not empty. It has been shown that by repeatedly applying the crown reduction, a linear sized kernel can be obtained in polynomial time [CFJ04]. Since determining whether a graph has a nontrivial crown decomposition is in P [ALS05], then crown reductions provide a new way of producing a $2k$ kernel for VERTEX COVER that is different from the Nemhauser-Trotter algorithm.

To apply crown decompositions we need to know when we can expect to find one. A very useful result in this regard appears in [CFJ04].

Lemma 1.3 *[CFJ04] If graph G has an independent set $I \subseteq V$ such that $|I| > |N(I)|$ then a non-trivial* crown decomposition (C, H, X) with $C \subseteq I$ can be found in $O(|V| + |E|)$ *time.*

If we are able to find a large enough independent set in a graph instance, and the problem we are solving admits a crown rule to reduce arbitrary instances, then the instance we are solving will certainly admit a crown decomposition and thus will be reduced by the crown rule.

Implementing crown rules to solve VERTEX COVER has proved an efficient way to solve the problem. In [AL+03] and [AC+04] Langston et al. present a comparison of performances of different algorithms for the VERTEX COVER problem. These experiments demonstrate that crown rules can be faster to solve VERTEX COVER than other techniques based on linear programming or network flow algorithms.

Crown decompositions can be modified to suit different problems. This has already become a useful tool to obtain significantly smaller (often linear) sized problem kernels. These modifications are known as *modeled crown decompositions*. In the following chapters we show examples of how to modify crown decomposition basic principles to address different problems: we find *double crowns* and *fat crowns* in Chapter 5, *fat-head crowns* in Chapter 6 and *k-spike crowns* in Chapter 8. These variations use an auxiliary graph to reduce the size of the instances and will be explained in detail in those chapters. Other modifications of the crown rule can be found in [FH+04] and [DFRS04].

1.6   OVERVIEW OF WORK

The work we present is divided into seven chapters. Each one of these chapters introduces a fixed-parameter algorithm to solve a different problem. The problems are MAX CUT, MAX LEAF SPANNING TREE, NONBLOCKER, s-STAR PACKING, EDGE-DISJOINT TRIANGLE PACKING, MAX INTERNAL SPANNING TREE and MINIMUM MAXIMAL MATCHING. Some of these problems had not been approached from a parameterized complexity viewpoint. Others had been studied before, but the *method of extremal structure* provides new insight and improves previous results.

The following results have been previously published: MAX CUT [P05], EDGE-DISJOINT TRIANGLE PACKING [MPS04], MAX INTERNAL SPANNING TREE [PS03] and s-STAR PACKING [PS04]. However, methodological differences, in some cases substantial, can be found between the published results and those presented in this thesis. We also correct several errors which appeared in those papers.

The *method of extremal structure* has been used to solve other graph problems such as VERTEX DISJOINT TRIANGLE PACKING [FH+04] as well as non-graph problems such as SET SPLITTING [DFR03].

2

MAX CUT

MOTIVATION

MAX CUT is a well studied problem both within and outside the field of parameterized complexity. The aim is to find a partition of the vertex set of a graph into two subsets with at least k edges having endpoints on different sides of the partition. As a decision problem, it was first proved NP-complete in 1976 [GJS76]. As an optimization problem, it is APX-complete [PY91], approximable within a factor of 1.1383 [GW95], and admits a PTAS if $|E| = \Theta(|V^2|)$ [AKK95]. We use the following natural parameterization of MAX CUT:

> MAX CUT
>
> Instance: A graph $G = (V, E)$ and a positive integer k
>
> Parameter: k
>
> Question: Is there a partition of the vertex set into two subsets A and B such that the size of the subset $E^* = E(A, B)$ is at least k?

The best known fixed-parameter algorithm for MAX CUT runs in time $\mathcal{O}(|E|+|V|+k \cdot 4^k)$ [MR99]. Fedin and Kulikov [FK02] give the current best known exact algorithm for the problem. The running time of this algorithm is $\mathcal{O}(poly(|E|) \cdot 2^{|E|/4})$.

This chapter presents an example of the *method of extremal structure* in its simplest form. Some of the reduction rules and structural claims in this chapter have been previously reported in [P05]. Overall, the results presented in this chapter simplify those of [P05].

REDUCTION RULES

First observe that if there exists a vertex $v \in V$ of degree at least k then, without loss of generality, v can be placed in A and all its neighbors in B. Hence the set $E^* = E(A, B)$ has size at least k, proving the following statement:

Proposition 2.1 *If there exists a vertex $v \in V$ of degree at least k, then (G, k) is a* YES-*instance for* MAX CUT.

Reduction Rule 1 *Let $u \in V$ be a vertex of degree one and let $v = N(u)$. Transform (G, k) into $(G \smallsetminus u, k - 1)$.*

Figure 2.1: Reduction Rule 1 for MAX CUT.

Soundness Lemma 1 Reduction Rule 1 is sound.

> ***Proof of Soundness Lemma 1.*** $(\Rightarrow:)$ Assume G is a graph which admits a bipartition of its vertex set into two subsets A and B with k edges in $E^* = E(A, B)$. Assume without loss of generality that v is in the subset A. By deleting u we reduce the number of edges in E^* by at most one as in the worst case v is in B and $(u, v) \in E^*$.
>
> $(\Leftarrow:)$ Conversely, assume G' is a graph with a bipartition of its vertex set into two subsets A' and B' with $k - 1$ edges in $E^{**} = E(A', B')$. Assume without loss of generality that v is in A'. Assume that we add the vertex u and the

edge (u, v) to G' to obtain G. By now placing u in B', we add one more edge to E^{**}. This constitutes a new instance with k edges in the cut between A' and B'. Hence, (G, k) is a Yes-instance for Max Cut. $\square$

Reduction Rule 2 *Let x, y and z be three vertices of degree two in G. Let $N(x) = \{u, y\}$, $N(y) = \{x, z\}$, $N(z) = \{y, v\}$. Let $u \neq v$. Let G' be the graph obtained by deleting y and z from G and adding the edge (x, v). Transform (G, k) into $(G', k-2)$.*

Figure 2.2: Reduction Rule 2 for Max Cut.

Soundness Lemma 2   Reduction Rule 2 is sound.

Proof of Soundness Lemma 2.

$(\Rightarrow:)$ Assume G is a graph which admits a bipartition of its vertex set into two subsets A and B with at least k edges in $E^* = E(A, B)$. Two cases arise:

- Both u and v are in the same side of the partition. Without loss of generality assume this side is A. In the worst case, the edges $(u, x), (x, y), (y, z)$ and (z, v) are all in E^*. Then, x and z are in B and y is in A. By deleting y and z three edges are lost from E^*. By adding (x, v) one is added. The result is an instance with at least $k - 2$ edges in E^*.

- If u and v are in opposite sides of the partition then, again without loss of generality, assume that $u \in A$ and $v \in B$. Since we are trying to

maximize the number of edges in the cut having all x, y and z in either A or B would never be part of an optimal solution. Thus, two cases arise:

- If x and z are in B and y is in A, then $(u, x), (x, y)$ and (y, z) are in E^*. By deleting y and z two edges are lost from E^*. By adding (x, v) no edges are added to E^* since both x and v are in B. The result is an instance with at least $k - 2$ edges in E^*.

- If x and z are in A and y is in B, $(z, v), (x, y)$ and (y, z) in E^*. By deleting y and z three edges from E^* are lost. By adding (x, v) we gain one edge to E^* since x is in A and v is in B. The result is also an instance with at least $k - 2$ edges in E^*.

$(\Leftarrow:)$ Assume that G' is a graph with a bipartition of its vertex set with at least $k - 2$ edges in E^*. When adding y and z back two cases arise:

- If $u, v \in V(G')$ are on the same side of the partition then, without loss of generality, assume that this side is A. If y is placed in A and x, z in B the edges (x, y), (y, z) and (z, v) are gained in E^*. In the worst case, if x was in B, (x, v) is lost. The result is an instance with two more edges in E^* and thus (G, k) is a YES-instance for MAX CUT.

- If $u, v \in V(G')$ are in opposite sides of the partition then, without loss of generality, assume that $u \in A$ and $v \in B$. The vertex x could be in either A or B. Assume first that x is in A. If y is placed in B and z in A, then the edges (x, y), (y, z) and (z, v) are in E^* and the edge (x, v) is lost. Thus (G, k) is a YES-instance for MAX CUT. If x is in B, then by placing y in A and z in B, the edges (x, y) and (y, z) are in E^* and thus (G, k) is a YES-instance for MAX CUT.

$\square$

Reduction Rule 3 *Let x, y and z be three vertices of degree two in G. Let $N(x) = \{u, y\}$, $N(y) = \{x, z\}$, $N(z) = \{y, u\}$. Let G' be the graph obtained by deleting x, y and z from G. Transform (G, k) into $(G', k - 4)$.*
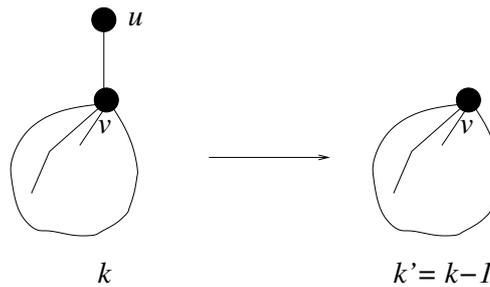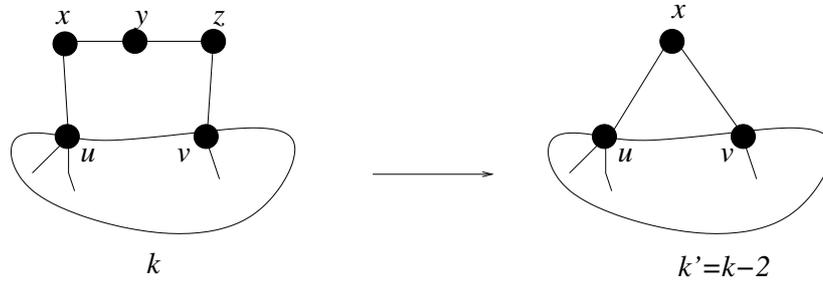

Figure 2.3: Reduction Rule 3 for MAX CUT.

Soundness Lemma 3 Reduction Rule 3 is sound.

Proof of Soundness Lemma 3.

($\Rightarrow$:) Suppose that (G, k) is a YES-instance of MAX CUT. Assume, without loss of generality, that $u \in A$. In the worst case, the edges (u, x), (x, y), (y, z) and (z, u) are in all E^*. Thus, x and z are in B and y is in A. By deleting x, y and z at most four edges are lost from E^*. Hence $(G', k-4)$ is a YES-instance for MAX CUT.

($\Leftarrow$:) We have to prove that by adding to G' vertices x, y and z, we will always obtain a solution with four more edges in E^*. Assume without loss of generality that u is in A. By placing x and z is in B and y is in A we gain four edges in E^*, namely $(u, x), (x, y), (y, z)$ and (z, u). Hence (G, k) is a YES-instance of MAX CUT. $\qquad\qquad\square$

Reduction Rule 4 *Let G be a connected graph. Let u be an articulation vertex in G and $G \smallsetminus u = X \cup Y$, with X and Y the two disjoint components arising when we delete u. Let $G' = G_1 \cup G_2$, where $G_1 = X \cup u'$ and $G_2 = Y \cup u''$ and u' and u'' are two new vertices such that $N(u') = N|_X(u)$ and $N(u'') = N|_Y(u)$. Transform (G, k) into (G', k).*
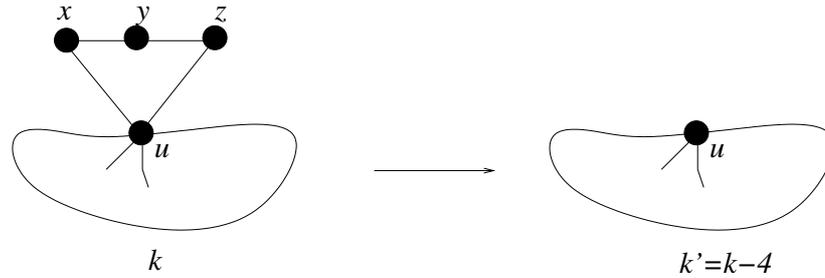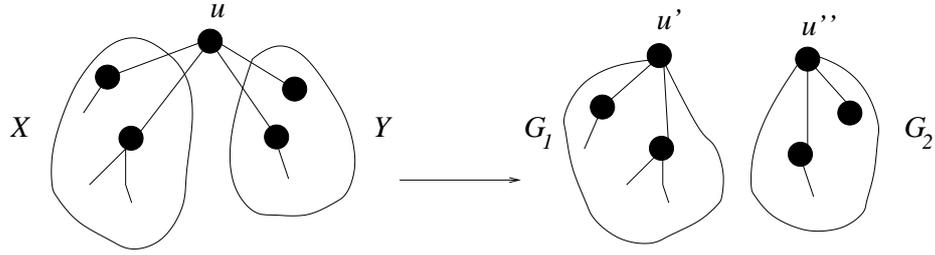
Figure 2.4: Reduction Rule 4 for MAX CUT.

Soundness Lemma 4 Reduction Rule 4 is sound.

Proof of Soundness Lemma 4.

($\Rightarrow$:) Assume that there exists a partition of $V(G) = A \cup B$ such that $E(A, B) = E^*$, $|E^*| \geq k$. Without loss of generality, assume that $u \in A$. Let $A' = A \smallsetminus u$ and $B' = B$. Make both u' and u'' be in A'. The two sets A' and B' form a bipartition of $G_1 \cup G_2$ such that $|E(A', B')| \geq k$ since no new edges are created and the ones previously in E^* are in $E(A', B')$. Thus, $(G_1 \cup G_2, k)$ is a YES-instance for MAX CUT.

($\Leftarrow$:) Suppose $(G_1 \cup G_2, k)$ is a YES-instance for MAX CUT. Let $V(G_1 \cup G_2) = A' \cup B'$ such that $|E(A', B')| \geq k$. Since u is split into u' and u'' when we merge them back into a single vertex two situations arise. Without loss of generality either:

- Both u' and u'' are in A'. Then making $A' = A$ and $B' = B$ and placing u in A will make (G, k) a YES-instance for MAX CUT.

- Vertex u' is in A' and u'' is in B'. Then, for every vertex w in G_2, if $w \in A'$, make $w \in B'$, and if $w \in B'$, make $w \in A'$. The instance $(G_1 \cup G_2, k)$ is still a YES-instance for MAX CUT but now u'' is in A' and the previous case applies.

$\square$

Reduction Rule 5 *Let u and v be two adjacent vertices in V of degree two. Let u and v be adjacent to the same vertex $w \in V$. Let $G' = G \smallsetminus \{u, v\}$. Transform (G, k) into $(G', k - 2)$.*
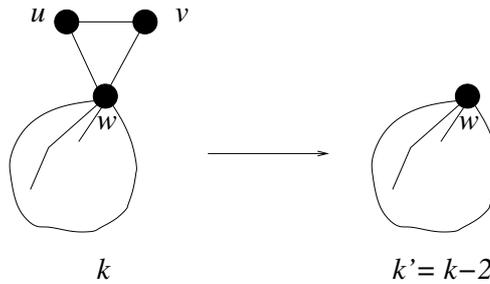
Figure 2.5: Reduction Rule 5 for MAX CUT.

Soundness Lemma 5 Reduction Rule 5 is sound.

Proof of Soundness Lemma 5.

Trivially follows from Reduction Rule 4.

$\square$

Reduction Rule 6 *Let u be a vertex of degree two in V. Let $N(u) = \{x, y\}$ and $(x, y) \in E$. Let $G' = G \smallsetminus \{u, (x, y)\}$, i.e., G' is the graph obtained by deleting the vertex u and the edge (x,y) from G. Transform (G, k) into $(G', k - 2)$.*

Soundness Lemma 6 Reduction Rule 6 is sound.

Proof of Soundness Lemma 6.

($\Rightarrow$:) Assume that (G, k) is a YES-instance of MAX CUT. Two cases arise:

- Both x and y are in the same side of the partition. By deleting u at most two edges from E^* are removed.
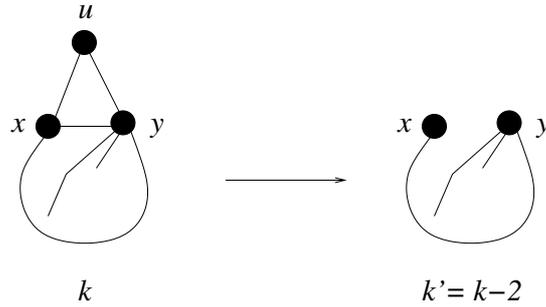
Figure 2.6: Reduction Rule 6 for MAX CUT.

- If x and y are in opposite sides of the partition then $(x, y) \in E^*$. The vertex u always has one adjacency with a vertex on the other side of the partition, which would add another element to the set E^*. By deleting both (x, y) and u the size of E^* is decreased by at most two.

($\Leftarrow$:) Assume that $(G', k - 2)$ is a YES-instance of MAX CUT. The vertices x and y are independent as we have deleted (x, y). Two cases arise depending on whether x and y are in the same or opposite sides of the partition and (G, k) is a YES-instance of MAX CUT.

- Both x and y are in the same side of the partition. By adding u and placing it in the opposite side, two edges are added to E^*, namely (u, x) and (u, y).

- If x and y are in opposite sides of the partition then the edge $(x, y) \in E^*$ adds one edge to E^*. No matter where u is placed it will add another element to the set E^*. Thus the size of E^* is increased by two and (G, k) is a YES-instance of MAX CUT.

$\square$

Reduction Rule 4 can potentially split a graph instance (G, k) into several biconnected components. We use the following result to prove Proposition 2.2.

Proposition 2.2 *Let G be a graph consisting of t disjoint connected components $\{G_1, G_2, \cdots G_t\}$.*
Let the graph G' be recursively constructed as follows:

- *For each $G_i \subset G$ select an arbitrary vertex $x_i \in G_i$.*

- *For each component G_i with $1 < i \leq t$, merge x_1 and x_i into a new vertex $x_1 \leftrightarrow x_i := x_1$. Make the resulting graph $G_{[x_1 \leftrightarrow x_i]} := G'$.*

Then, (G, k) is a YES-instance for MAX CUT if and only if (G', k) is a YES-instance for
MAX CUT.
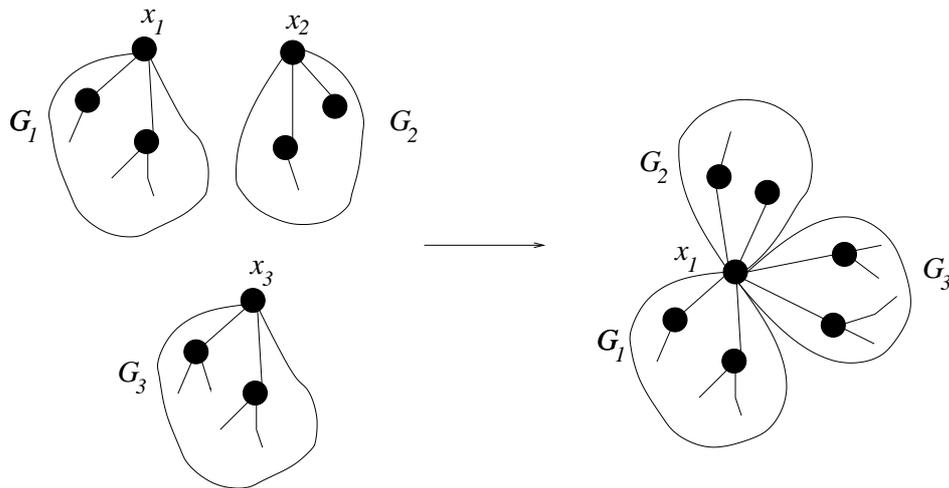
Figure 2.7: Example of construction of G' when $t = 3$.

Proof of Proposition 2.2.

$(\Rightarrow:)$ Assume that (G, k) is a YES-instance of MAX CUT, i.e., there exists a partition of
$V(G) = A \cup B$ such that $E(A, B) = E^*$, $|E^*| \geq k$. Assume that x_1 is in A. Then, we
can always find an indexing such that there exists an index t' such that $1 \leq t' \leq t$, for all
$i \leq t'$ $x_i \in G_i$ is in A and for all $t' < i \leq t$ $x_i \in G_i$ is in B. Then:

- For all $i \leq t'$ make $A' = A$ and $B' = B$.

- For all $t' < i \leq t$, for each vertex w in G_i, if $w \in A$, make $w \in B'$, and vice versa.

All vertices x_i are in A'. The number of edges in $E(A, B)$ is the same as the number of edges in $E(A', B')$. Hence, (G', k) is a YES-instance for MAX CUT.

($\Leftarrow$:) Assume that there exists a partition of $V(G') = A' \cup B'$ such that $E(A', B') = E^{**}$, $|E^{**}| \geq k$. The vertex x_1 is an articulation vertex in G'. Making $A' = A$ and $B' = B$ will provide a partition of G with at least k edges in $E(A, B)$ as the total number of edges remains the same. Thus, (G, k) is a YES-instance of MAX CUT. $\qquad\square$

KERNELIZATION AND BOUNDARY LEMMAS

In our analysis we only use Reduction Rules 1 to 5 and an instance is considered *reduced* when these rules are no longer applicable. Reduction Rule 6 is included since it may be of interest from a practical point of view.

Lemma 2.1 *(**Kernelization Lemma**) If an instance (G, k) is reduced and has at least k vertices or $2k$ edges, then it is a YES-instance of MAX CUT.*

To prove the kernelization lemma we need to prove Lemma 2.2 and Lemma 2.3, the boundary lemmas.

Lemma 2.2 *(**Boundary Lemma - Vertices**) Let (G, k) be a reduced instance of MAX CUT. Let (G, k) be a YES-instance and $(G, k+1)$ be a NO-instance of MAX CUT. Then, the number of vertices in G is at most k.*

Proof of Lemma 2.2. (By minimum counterexample) Let (G, k) be reduced, and a YES-instance for MAX CUT. Let $(G, k + 1)$ be a NO-instance of MAX CUT. Assume for contradiction that $|V(G)| \geq k + 1$.

We can assume, without loss of generality, that G consists only of one biconnected component, since if there were more components, we can process each of them separately and use Proposition 2.2 to splice all disjoint biconnected components together.

Consider, as a witness of the fact that (G, k) is a YES-instance of MAX CUT, a spanning tree T in G. Since this spanning tree has at least $k + 1$ vertices, it has at least k edges. Let's denote the edge set of T, $E(T) = E^*$. We construct a partition of the vertices in $V = A \cup B$ such that all edges in E^* are in $E(A, B)$. Such a tree can be constructed as follows. Choose any leaf v in T and place v in A. Consider the neighbor w of v in T and place it in B. The edge (u, v) is in E^*. Recursively assign all neighbors in T of a vertex in A to B and vice versa. For each one of the vertices placed in A (respectively, B) we are augmenting the number of elements in E^* by one. Repeat this process until all vertices are placed into either A or B.
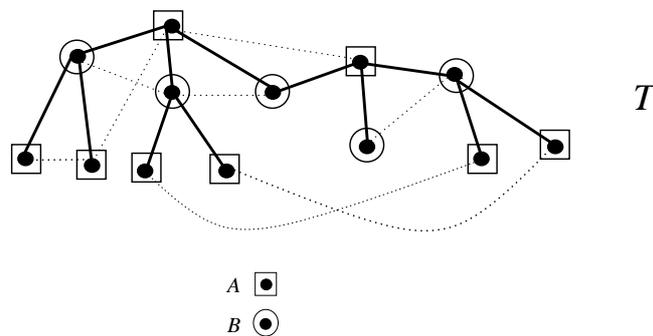
Figure 2.8: Example of witness tree T with partition of vertex set into A and B.

The graph G has more edges which are not in T. We will call these edges *ghost edges*[1]. We call *ghost neighbors* two vertices adjacent via a ghost edge.

We set the following inductive priority:

⋆ *Among all counterexamples, consider those minimizing the number of leaves in T.*

---

[1] In the figures ghost edges are represented by dotted lines.

Structural Claims

Claim 1 *G has $k + 1$ vertices.*

Proof of Claim 1. Assume G had more than $k + 1$ vertices. Consider the spanning tree T in G. Since T has at least $k + 2$ vertices then it has at least $k + 1$ edges. By construction, all these edges are in E^*. Hence, if G has more that $k + 1$ vertices then $(G, k + 1)$ is a YES-instance of MAX CUT, contradicting the hypotheses of Lemma 2.2. □

We have proved that the witness spanning tree T has size $k + 1$ and we can partition the vertex set $V = (A \cup B)$ so that all edges in T are in E^*.

Claim 2 *There are no ghost edges between vertices of A and vertices of B.*

Proof of Claim 2. Since T has already k edges, any ghost edge between A and B would make $(G, k + 1)$ a YES-instance for MAX CUT, contradicting the hypotheses of Lemma 2.2. □

Claim 3 *All leaves in T have degree two in G.*

Proof of Claim 3. By Reduction Rule 1 there are no vertices of degree one in G. For contradiction suppose that there exists a leaf v in T with degree at least three in G. Let the neighbor of v in T be vertex u. Without loss of generality assume that v is in A. By Claim 2 there are no ghost edges that link v to vertices in B and thus v must have at least two ghost neighbors $x, y \in A$. If we now move v to B then we lose the edge (u, v) from E^* but gain both (v, x) and (v, y) in E^*. Thus $(G, k + 1)$ is a YES-instance for MAX CUT, contradicting the hypotheses of Lemma 2.2. □
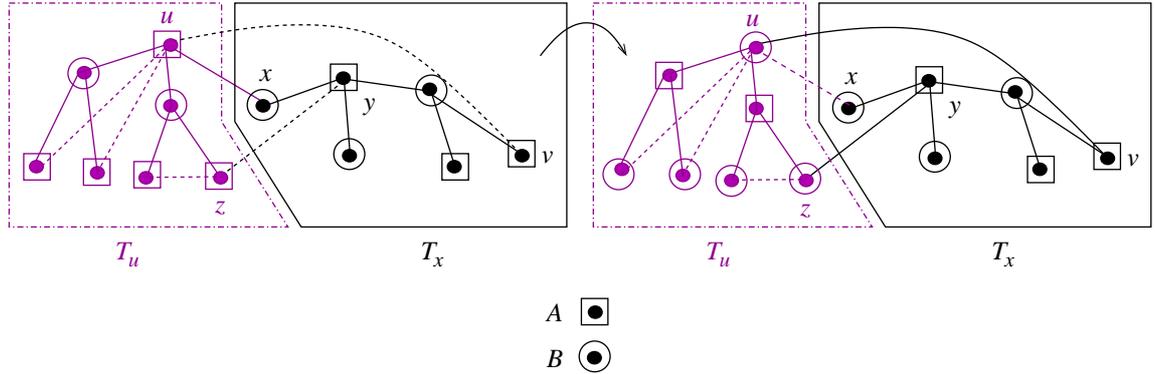
Figure 2.9: Example of transformation in Claim 4.

Claim 4 *There are no ghost edges between leaves and internal vertices in T.*

Proof of Claim 4. Let v be a leaf of T. Without loss of generality assume that v is in A. By Claims 2 and 3, v has a unique ghost neighbor $u \in A$. Let P be the path from u to v in T and let x be the neighbor of u in this path. Since the graph is reduced by Reduction Rule 4 then u cannot be an articulation vertex in G, and thus there must exist a ghost edge going from the right subtree T_x to the left subtree T_u of the edge (u, x). Let this edge be (y, z) with $y \in T_x$ and $z \in T_u$. We can now modify the solution witnessed by T by exchanging the roles of A and B in all vertices in T_u. By doing this we lose (u, x) from E^* but we gain (u, v) and (y, z) in E^*. Note that the we have modified T to obtain another subgraph that is no longer a tree, but it certainly contains a cut with $k + 1$ edges.

Thus, if there existed a ghost edge between a leaf and an internal vertex in T, $(G, k + 1)$ would be a YES-instance for MAX CUT, contradicting the hypotheses of Lemma 2.2. □

Claim 5 *Two leaves u and v in T which are adjacent in G cannot have the same parent w in T.*

Proof of Claim 5. Since u and v must have degree two in G by Claim 3, if they are adjacent and have the same parent in T, then they form a dangling triangle in G. This type of triangles is handled by Reduction Rule 5. Hence, if u and v have the same parent w in T, the instance (G, k) is not reduced under Reduction Rule 5. This contradicts the hypotheses of Lemma 2.2. $\square$

Claim 6 *There are no ghost edges between leaves in T.*

Proof of Claim 6. Let u and v be two leaves in T and assume for contradiction that there is a ghost edge between them. Let P be the path from u to v in T. Let x be the neighbor of u in P, and let y be the other neighbor of x in P. By Claim 2, u and v must be in the same side of the partition. By Claim 5, x cannot be the parent vertex of both u and v. Without loss of generality assume that u and v are in A. The vertex x is then in B by hypothesis of the witness structure. Since x is internal to T, two cases arise:

- Vertex x has degree at least three in T. We can modify T by changing u from A to B, losing the edge (x, u) from T (and thus from E^*) and gaining (u, v) in the modified spanning tree (and thus in E^*). Now vertex u is a leaf in the modified spanning tree, x is still internal, as it has degree at least two, but v is no longer a leaf. We have constructed a new spanning tree with $k + 1$ vertices such that all edges in the tree are in E^*, but the number of leaves is decreased by one (see Figure 2.10). This contradicts the inductive priority stating that the number of leaves in T was minimum.

- Vertex x has degree two in T. Then since u and v have degree two in G (by Claim 3) and the graph is reduced by Reduction Rule 2 and Reduction Rule 3, x cannot have degree two in G. Hence x must have a ghost neighbor z in T and since x is in B this neighbor must also be in B (by Claim 2). We can modify A and B by changing u from A to B.
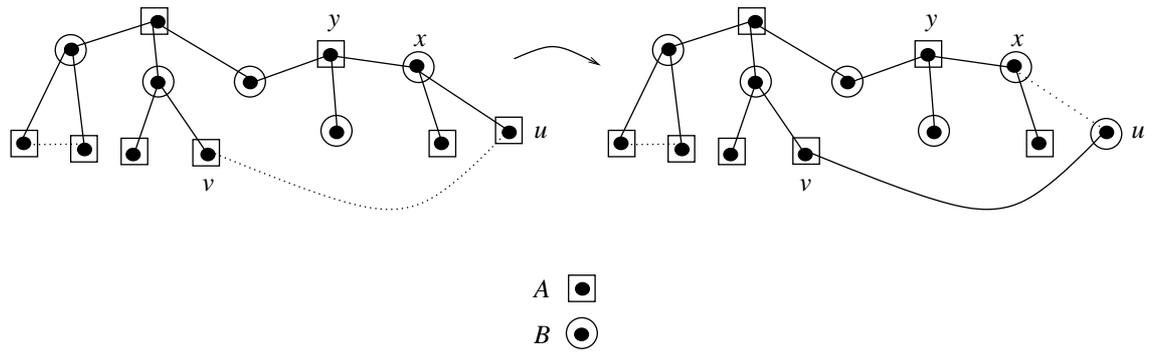
Figure 2.10: Example of transformation in Claim 6 when x has degree at least three in T.

This way the edge (x, u) is no longer in E^*, but (u, v) is now in E^*. If x is moved to A, the edge (x, y) is no longer in E^* but (x, u) and (x, z) are now in E^* (see Figure 2.11).
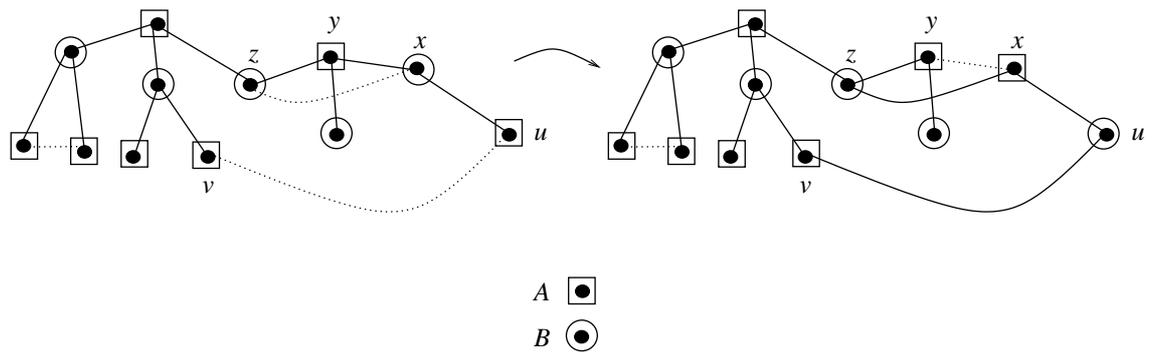
Figure 2.11: Example of transformation in Claim 6 when x has degree two in T.

The witness solution is no longer a tree, but it contains a cut with $k + 1$ edges. Thus $(G, k + 1)$ is a YES-instance for MAX CUT, contradicting the hypotheses of Lemma 2.2.

□

We have proved that if G has $k + 1$ vertices then the leaves of T cannot have ghost neighbors in either the leaves of T or the internal vertices of T. By Reduction Rule 1

there are no vertices of degree one in G and the leaves in T must have neighbors. We have reached a contradiction. Therefore, if (G, k) is reduced, (G, k) is a YES-instance and $(G, k + 1)$ is a NO-instance for MAX CUT, then $|V(G)| \leq k$. $\square$

Lemma 2.3 *(**Boundary Lemma - Edges**) Let (G, k) be a reduced instance of* MAX CUT. *Let (G, k) be a* YES-*instance and $(G, k + 1)$ is a* NO-*instance of* MAX CUT. *Then, the number of edges in G is less than $2k$.*

Proof of Lemma 2.3. (By minimum counterexample) Assume that (G, k) is reduced and admits a partition of $V(G)$ into two subsets A and B with $E^* = E(A, B)$ such that $|E^*| = k$. Let $(G, k + 1)$ be a NO-instance of MAX CUT. Assume for contradiction that $|E(G)| \geq 2k$.

Consider, as a witness of the fact that (G, k) is a YES-instance, a partition of the vertex set $V = (A, B)$ with k edges $(u, v) \in E$ where $u \in A$ and $v \in B$. These edges constitute E^*.

The graph may have edges which are not in E^*. These edges are termed *ghost edges*. Each element $u \in A$ has two degrees associated with it, namely $deg_{|A}(u)$ and $deg_{|B}(u)$. Note that $deg_{|A}(u)$ is the degree of u in $\langle A \rangle$. By Proposition 2.1 and the fact that $(G, k + 1)$ is a NO-instance, $deg_{|A}(u) + deg_{|B}(u) = deg(u) \leq k$. Similarly, each element $v \in B$ has two degrees associated with it $deg_{|A}(v)$ and $deg_{|B}(v)$ where $deg_{|B}(v)$ is the degree of v in $\langle B \rangle$ and $deg_{|A}(v) + deg_{|B}(v) = deg(v) \leq k$.

Note that since (G, k) is a YES-instance and $(G, k + 1)$ is a NO-instance,

$$\sum\nolimits_{u \in A} deg_{|B}(u) = \sum\nolimits_{v \in B} deg_{|A}(v) = k.$$

We set the following inductive priority:

$\star$ *Among all counterexamples, choose those minimizing $|A|$.*

Claim 1 *For all $u \in A$, $deg_{|A}(u) < deg_{|B}(u)$.*

Proof of Claim 1. Assume that there exists a vertex u in A contradicting Claim 1. Moving u to B causes the loss of $deg_{|B}(u)$ edges from E^* and the gain of $deg_{|A}(u)$ in E^*. Two cases arise:

- If $deg_{|A}(u) > deg_{|B}(u)$, then $(G, k+1)$ is a YES-instance for MAX CUT, contradicting the hypotheses of Lemma 2.3.

- If $deg_{|A}(u) = deg_{|B}(u)$, then moving u to B implies that (G, k) is still a YES-instance for MAX CUT as the number of edges lost from E^* is the same as the number of edges gained. However, the size of A has decreased, contradicting the minimality of A ensured by the inductive priority.

□

Claim 2 *For all $v \in B$, $deg_{|A}(v) \geq deg_{|B}(v)$.*

Proof of Claim 2. If $deg_{|A}(v) < deg_{|B}(v)$, moving v to A causes the loss of $deg_{|A}(v)$ edges from E^* and the gain of $deg_{|B}(v)$ edges. Thus, $(G, k+1)$ is a YES-instance for MAX CUT, contradicting the hypotheses of Lemma 2.3. □

Claim 3 *The number of ghost edges in the subgraph induced by A is less than $k/2$.*

Proof of Claim 3. By Claim 1, $\sum_{u \in A} deg_{|A}(u) < \sum_{u \in A} deg_{|B}(u) = k$. By Euler's theorem, the sum of the degrees in a graph is exactly twice the number of edges. Thus,

$$|E(A)| = \frac{\sum_{u \in A} deg_{|A}(u)}{2} < \frac{\sum_{u \in A} deg_{|B}(u)}{2} = \frac{k}{2}.$$

□

Claim 4 *The number of ghost edges in the subgraph induced by B is at most $k/2$.*

Proof of Claim 4. By Claim 2, $\sum_{v \in B} deg_{|B}(v) \leq \sum_{v \in A} deg_{|A}(v) = k$. By Euler's theorem, the sum of the degrees in a graph is exactly twice the number of edges. Thus,

$$|E(B)| = \frac{\sum_{v \in B} deg_{|B}(v)}{2} \leq \frac{\sum_{v \in B} deg_{|A}(v)}{2} = \frac{k}{2}.$$

$\square$

The total number of edges in G is $E(A, B)$ plus the ghost edges in $\langle A \rangle$ and $\langle B \rangle$. Thus,

$$|E(G)| = |E^*| + |E(A)| + |E(B)| < k + \frac{k}{2} + \frac{k}{2} = 2k.$$

$\square$

Now we prove Lemma 2.1. This lemma states that if (G, k) is a reduced instance and has at least k vertices or $2k$ edges, then it is a YES-instance of MAX CUT.

Proof of Lemma 2.1. (***Kernelization Lemma***) Assume in contradiction to the stated Lemma 2.1 that there exists a graph instance (G, k) such that $|V(G)| > k$ or $|E(G)| > 2k$ but (G, k) is a NO-instance for MAX CUT.

Let $k' < k$ be the largest k' for which (G, k') is a YES-instance. By Lemma 2.2 we know that $|V(G)| \leq k' < k$. This contradicts the assumption.

Let $k' < k$ be the largest k' for which (G, k') is a YES-instance. By Lemma 2.3 we know that $|E(G)| \leq 2k' < 2k$. This contradicts the assumption. $\square$

ALGORITHM AND RUNNING TIME ANALYSIS

In the previous section we proved a kernel for the problem of size k in the number of vertices and $2k$ in the number of edges. In Table 2.1 we provide an algorithm to solve MAX CUT.

Step 1.   Apply reduction rules 1 to 5 to (G, k) until the
          instance is reduced.
Step 2.   If $|E(G)| \geq 2k$ or $|V(G)| \geq k$ then answer YES and
          halt.
Step 3.   If $|V(G)| < \frac{k}{2}$ then try all possible partitions
          of the vertex set into two subsets A and B and
          count the number of edges between them. If
          there are more than k answer YES.
Step 4.   Else if $|V(G)| \geq \frac{k}{2}$ then find a maximum cut in the
          kernel using the algorithm in [FK02]. If the
          cut has size greater than k answer YES.
Step 5.   If the answer is YES in Step 3 or Step 4, an-
          swer YES and halt.
Step 6.   Else answer NO and halt.

Table 2.1: FPT algorithm to solve MAX CUT.

Lemma 2.4 MAX CUT *can be solved in time* $\mathcal{O}^*(1.414^k)$.

Proof of Lemma 2.4. We will analyze each step in the algorithm separately.

Step 1. Reduction Rule 1 can be performed in linear time in the size of V as there are at most $|V|$ vertices of degree one, and deleting them can be done in constant time. Reduction Rule 2 is concerned with finding vertices of degree two and checking if both their neighbors have also degree two. This operation can be performed in time $\mathcal{O}(|V|)$ as the number of vertices of degree two can be at most $|V|$ and the transformation can be done in constant time. Similarly, Reduction Rules 3 and 5 can be performed in time $\mathcal{O}(|V|)$.

In Reduction Rule 4 we must find the articulation vertices in G. There are at most $|V|$ such vertices and each one can be found in time $\mathcal{O}(|V|^2)$. The running time of this reduction rule is thus $\mathcal{O}(|V|^3)$.

The running time of Step 1 is bounded by sum of the running times of the four reduction rules, $\mathcal{O}(|V|^3)$.

Step 2. This step can be performed in time $\mathcal{O}(k)$.

Step 3. The total number of different partitions in a graph with n vertices is $2^{n-1} - 1$. Counting the number of vertices can be done in time $\mathcal{O}(k)$ since the graph is reduced. If this number is at most $k/2$ we try all partitions. Checking how many edges exist in the bipartite subgraph induced by each partition of $V(G)$ into $A \cup B$ can always be performed in time $\mathcal{O}(k)$ since there are only $2k$ edges in the graph. Thus the running time of Step 3 is $\mathcal{O}(k \cdot 2^{k/2}) = \mathcal{O}^*(1.414^k)$.

Step 4. Applying the exact algorithm proposed by Fedin and Kulikov [FK02] to the kernel leads to a running time of $\mathcal{O}^*(1.414^k)$, as the running time of their algorithm is $\mathcal{O}(poly(|E|) \cdot 2^{|E|/4})$, and by Step 2 we know that the graph has at most $2k$ edges.

Again, checking if the cut has size greater than k can be performed in time $\mathcal{O}(k)$ since there are only $2k$ edges in the graph.

Thus the running time of Step 4 is $\mathcal{O}(poly(|E|) \cdot 2^{|E|/4}) = \mathcal{O}^*(1.414^k)$.

Step 5 and Step 6. This last part of the algorithm can be performed in constant time.

Thus, the total running time of this algorithm is the minimum of the running times of Steps 3 and 4 and MAX CUT can be solved in time $\mathcal{O}^*(1.414^k)$. $\square$

MAX LEAF

MOTIVATION

The problem of finding a spanning tree with many leaves in a graph is of practical interest in areas such as network design [CWY00]. The parameterized version of the problem, where we take as a parameter the number of leaves is defined as follows:

> MAX LEAF SPANNING TREE
>
> Instance: A connected graph $G = (V, E)$ and a positive integer k
>
> Parameter: k
>
> Question: Does G have a spanning tree T with at least k leaves?

As a classical decision problem, MAX LEAF SPANNING TREE was first proved NP-complete by Garey and Johnson [GJ79]. It is APX-complete [GMM97] and approximable within a factor of 2 [S98]. The best known parameterized algorithm to date achieves a kernel size of $4k$ and a running time of $\mathcal{O}^*(9.49^k)$ [BBW03].

There are many related problems about spanning trees that maximize an objective function. In Chapter 7 we will consider the parameterized problem of finding a tree with at least k internal vertices, proving that it is in FPT. In [GMM94] it is shown that the problems of finding a spanning tree that has maximum sum of the distances between all pairs of vertices, or maximum sum of the distances from a specified root, are not in PTAS unless P = NP.

The proof of Reduction Rule 1 and Structural Claims 1- 7 in this chapter were first sketched in [FM+00].

REDUCTION RULES

If T is a spanning tree for G, the set of internal vertices of T is denoted I and the set of leaves of T is denoted L.

Reduction Rule 1 *Let u and v be two adjacent vertices of degree greater than one in G and $(u,v) \in E(G)$ be a bridge in G. Let $G' = G_{[u \leftrightarrow v]}$, the graph obtained by merging u and v into a new vertex $u \leftrightarrow v$. Transform (G,k) into (G',k).*

Figure 3.1: Reduction Rule 1 for MAX LEAF SPANNING TREE.

Soundness Lemma 1 Reduction Rule 1 is sound.

Proof of Soundness Lemma 1.

First, we must point out that if (u,v) is a bridge in G and both u and v have degree greater than one, they must be internal to any spanning tree T in G. Otherwise the spanning tree would not be connected. Similarly, an articulation vertex in G is always internal to any spanning tree in G, as otherwise the spanning tree would not connect the two components which the articulation vertex is separating.

($\Rightarrow$:) Since u and v are internal to T, we can construct a tree T' for G' making the vertex $u \leftrightarrow v$ be internal to the T' and leaving the rest of the tree as in T. The number of leaves remains the same.

($\Leftarrow$:) Let T' be a spanning tree with k leaves in $G' = G_{[u \leftrightarrow v]}$. The articulation vertex $u \leftrightarrow v$ in G' is an internal vertex of T'. If we expand this articulation vertex to create G, we can construct a new tree T by adding the edge (u, v) to T, making u and v internal vertices. The number of leaves remains the same. $\square$

Reduction Rule 2 *Let u and v be two adjacent vertices of degree two in G, such that $(u, v) \in E(G)$ is not a bridge. Let $N(u) = \{v, x\}$ and $N(v) = \{u, y\}$. Let $G' = G \smallsetminus (u, v)$. Transform (G, k) into (G', k).*

Figure 3.2: Reduction Rule 2 for MAX LEAF SPANNING TREE.

Soundness Lemma 2 Reduction Rule 2 is sound.

Proof of Soundness Lemma 2.

($\Leftarrow$:) Let (G', k) be a YES-instance. Let T' be a spanning tree in G' with at least k leaves. After adding the edge (u, v) to $E(G)$, T' will also be a spanning tree for G with at least k leaves, as adding an edge will never decrease the number of leaves in an existing spanning tree for G. Thus, (G, k) is also a YES-instance for MAX LEAF SPANNING TREE. Note that this argument is valid whether $x = y$ or $x \neq y$.

($\Rightarrow$:) This part of the proof requires a case analysis according to the roles of u and v in a spanning tree T for G. First we will assume that $x \neq y$.

1. Vertices u and v are both leaves in T. In this case deleting (u, v) has no effect on the number of leaves in T.

2. Vertex u is internal to T and v is a leaf of T. Since u has degree 2, then the edge (u, v) is in T. In this case deleting (u, v) will make u a leaf for a new tree T' in G'. Vertex v can be made a leaf of T' by including the edge (v, y) in T', thus not decreasing the total number of leaves. The instance (G', k) is therefore a YES-instance.

3. Vertices u and v are both internal to T. Thus, the edge (u, v) belongs to T. Since (u, v) is not a bridge, there is at least one path in G to get from u to v which does not use (u, v). As T is a tree not all edges in this path are part of T so there exist two vertices u' and v' such that the edge between them is not in $E(T)$. If we delete (u, v) from T and add (u', v') to T we have a new spanning tree. In the worst case both u' and v' were leaves in T and thus we lose two leaves. We gain two leaves as u and v have now become leaves. Thus, $(G \smallsetminus (u, v), k)$ is a YES-instance for MAX LEAF SPANNING TREE.

The proof in the case of $x = y$ is identical in Cases 1 and 2. The last case is not applicable as it is not possible that u and v are both internal vertices to T when $x = y$ (otherwise there would be a cycle in T).

$\square$

Reduction Rule 3 *Let u be a vertex of degree one in G. Let $N(u) = w$ and $N\big(N(w)\big) \smallsetminus \{w\} \neq \varnothing$. Let G' be the graph obtained by deleting u from G and making a clique out of $N(w)$, i.e., for each pair of vertices $x, y \in N(w), x \neq y$, we add (x, y) to $E(G)$ if the edge $(x, y) \notin E(G)$. Transform (G, k) into (G', k).*
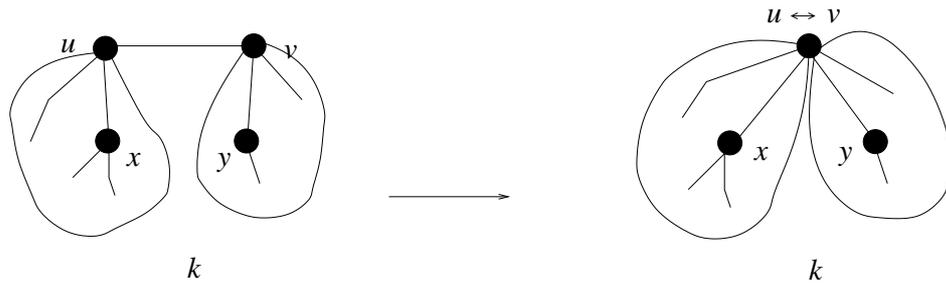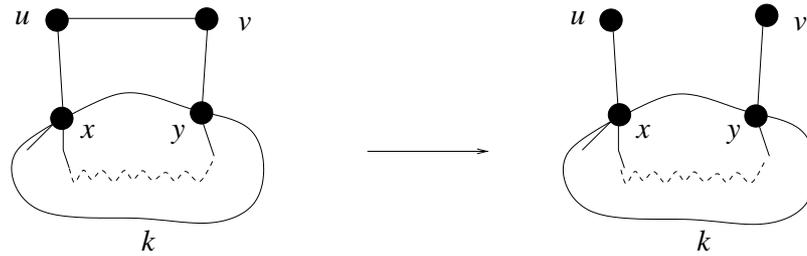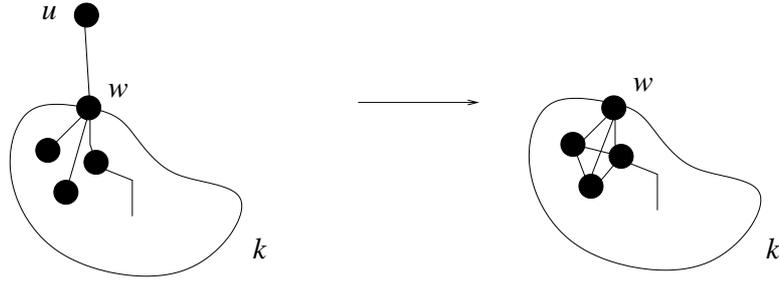
Figure 3.3: Reduction Rule 3 for MAX LEAF SPANNING TREE.

Soundness Lemma 3 Reduction Rule 3 is sound.

Proof of Soundness Lemma 3.

($\Rightarrow$:) Assume (G, k) is a YES-instance of MAX LEAF SPANNING TREE. Assume T is a spanning tree with k leaves in G. A vertex of degree one must be a leaf in T. Thus w is internal to T. If we delete u and make a clique out of all the neighbors of w then in the worst case we will be losing one leaf of T. We must show that by making a clique out of all neighbors of w we always gain a leaf back. Two possibilities arise:

- If w has degree two in T, then by deleting u, vertex w becomes a leaf in the new spanning tree T' of G'. The total number of leaves remains the same.

- If w has degree greater than two in T, there exists a vertex v in $N(w)$ also internal to the tree since $N(N(w)) \smallsetminus \{w\} \neq \varnothing$. Let $N_T(w) = \{u, v, v_1, v_2, \ldots, v_c\}$. The edges $\{(v, v_1), (v, v_2), \ldots, (v, v_c)\}$ are not in T as otherwise we would have cycles. Let T' be a spanning tree for G' constructed as follows. The tree T' is equal to T for all edges in T except for the edges (w, v_i), $i = 1, \ldots, c$. In this case we remove (w, v_i) from $E(T)$ and add (v, v_i) to $E(T')$. By doing this, we create a new leaf out of w without losing any of the leaves we had previously in T. The

tree T' has k leaves and thus (G', k) is a YES-instance for MAX LEAF SPANNING TREE.
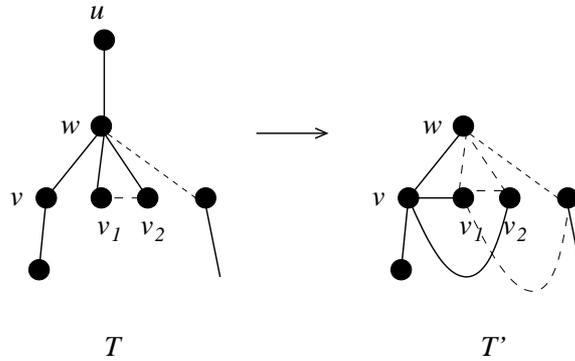
Figure 3.4: Transformation from T to T' after deleting vertex u.

($\Leftarrow$:) Suppose (G', k) is a YES-instance of MAX LEAF SPANNING TREE. Let T' be a spanning tree with k leaves in G'. If we add the vertex u to G', in the worst case we would be appending u to an existing leaf of T'. We lose one leaf in T' but u becomes a leaf in T, a spanning tree for G. If u is adjacent to an internal vertex in T' then we make u a leaf of T, constructing a spanning tree for G with $k + 1$ leaves. In either case the number of leaves does not decrease and (G, k) is a YES-instance for MAX LEAF SPANNING TREE. $\square$

An instance (G, k) of MAX LEAF SPANNING TREE is considered *reduced* when Reduction Rules 1, 2, and 3 can no longer be applied.

In the last section of this chapter we provide an algorithm to solve MAX LEAF SPANNING TREE that makes use of a *catalytic vertex*. A catalytic vertex is a vertex of degree at least three in G which we force to be internal in the spanning tree. We performapply this algorithm to an instance which is first reduced and then *catalyzed*, i.e., a reduced instance of MAX LEAF SPANNING TREE where a vertex of degree at least three is designated as catalytic. Hence, here we define the corresponding annotated problem:

ANNOTATED MAX LEAF SPANNING TREE

Instance: A connected graph $G = (V, E)$ with a *catalytic* vertex $r \in V$ of

              degree at least three, and a positive integer k

Parameter: k

Question: Does G have a spanning tree T with at least k leaves such that $r \in I$?

Lemma 3.1 shows how to *de-catalyze* an instance of ANNOTATED MAX LEAF SPANNING TREE, i.e., how to transform an instance (G, r, k) with catalytic vertex r of ANNOTATED MAX LEAF SPANNING TREE into an instance (G', k') of MAX LEAF SPANNING TREE.

Lemma 3.1 *(De-catalyzation Rule) Let (G, r, k) be an instance of* ANNOTATED MAX LEAF SPANNING TREE *with catalytic vertex r. Let G' be the graph obtained by adding a vertex x and the edge (r, x) to G. Then, (G, r, k) is a* YES-*instance for* ANNOTATED MAX LEAF SPANNING TREE *if and only if $(G', k + 1)$ is a* YES-*instance for* MAX LEAF SPANNING TREE.

> ***Proof of Soundness Lemma 3.1.*** $(\Rightarrow:)$ Let (G, r, k) be a YES-instance of ANNOTATED MAX LEAF SPANNING TREE with catalytic vertex $r \in I$. In the instance $(G', k' = k + 1)$ constructed as described above, x will be a new leaf adjacent to r in the tree. Thus, $(G', k + 1)$ is a YES-instance for MAX LEAF SPANNING TREE.
>
> $(\Leftarrow:)$ Conversely, assume $(G', k + 1)$ is a YES-instance of MAX LEAF SPANNING TREE. Since x is always a leaf in any spanning tree in G', its neighbor r will always be internal. Thus, when we remove x, its neighbor r can be made a catalytic vertex without losing any information about G. As we have deleted a leaf, (G, r, k) is a YES-instance for ANNOTATED MAX LEAF SPANNING TREE. $\square$

Observe that the de-catalyzation rule is only used once. This is important, since this is a transformation which actually enlarges the instance.

KERNELIZATION AND BOUNDARY LEMMAS

We denote as (G, r, k) the instance of ANNOTATED MAX LEAF SPANNING TREE obtained from (G, k) of MAX LEAF SPANNING TREE by choosing a catalytic vertex. We define a *reduced* instance of ANNOTATED MAX LEAF SPANNING TREE as a reduced instance of MAX LEAF SPANNING TREE where a vertex r of degree at least three has been designated as catalytic. Without loss of generality, we say that (G, r, k) is *reduced* under Reduction Rule 1 if (G, k) is reduced under Reduction Rule 1. Similarly we define being reduced under Reduction Rules 2 and 3.

Lemma 3.2 *(**Kernelization Lemma**) Let G a connected graph with a catalytic vertex $r \in I$. If (G, r, k) is reduced and G has at least $5.75k$ vertices, then (G, r, k) is a YES-instance for ANNOTATED MAX LEAF SPANNING TREE.*

To prove the kernelization lemma we need to prove Lemma 3.3, the boundary lemma.

Lemma 3.3 *(**Boundary Lemma**) Let G be a connected graph with a catalytic vertex $r \in I$. Let (G, r, k) be reduced. Suppose that (G, r, k) is a YES-instance and that $(G, r, k+1)$ is a NO-instance for ANNOTATED MAX LEAF SPANNING TREE. Then the number of vertices of G is less than $5.75k$.*

Proof of Lemma 3.3. (By minimum counterexample)

By hypothesis of minimum counterexample (G, r, k) is a YES-instance and $(G, r, k+1)$ is a NO-instance. Hence, there exists a tree T in G, not necessarily spanning, with k leaves. This tree will act as a witness that (G, r, k) is a YES-instance.

It is important to note that any such tree T can be inductively extended to construct a spanning tree with at least k leaves. If the vertex v we are adding is adjacent to an internal vertex u in T, we add the edge (u, v) to the new spanning tree, constructing a tree with

one more leaf. If v is only adjacent to leaves in T we choose one of them, w, and add the edge (v, w) to the new spanning tree. The new tree we have constructed in this case has the same number of leaves as T.

All vertices not in T are termed *outsiders*, and we denote the outsider set by O. In T we distinguish three types of vertices: L, the *leaves*; I, the *internals*; and I_B, the *branches*, those internal vertices having more than two tree-edges incident to them. Let the catalytic vertex $r \in I$ be the *root* of T. Let $d(v, r)$ be the distance in T between any vertex $v \in T$ and the root vertex r.
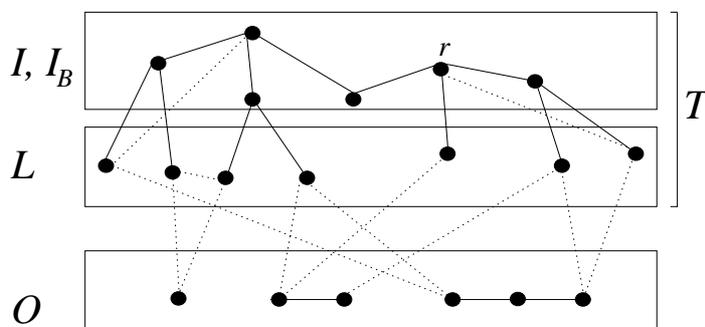
Figure 3.5: Example of witness tree T.

We set the following inductive priorities:

⋆ *Among all witnesses, consider those minimizing $|T|$.*

⋆ *Among all witnesses subject to the previous inductive priority, consider those minimizing $\sum_{l \in L} d(l, r)$.*

Structural Claims

Claim 1 *Every vertex in L has at most one neighbor in O.*

> *Proof of Claim 1.* If a leaf $u \in L$ has two neighbors v, w in O then we could extend T by making u an internal node to the tree and v and w leaves. The

instance $(G, r, k + 1)$ would therefore be a YES-instance for ANNOTATED MAX LEAF SPANNING TREE, contradicting our hypothesis.   □

Claim 2 *No vertex in I has any neighbors in O.*

Proof of Claim 2. If there is a vertex $u \in I$ with a neighbor v in O then we can make v a leaf of T and thus $(G, r, k + 1)$ is a YES-instance for ANNOTATED MAX LEAF SPANNING TREE, contradicting our hypothesis.   □

Claim 3 *There are no vertices of degree greater than two in $\langle O \rangle$.*

Proof of Claim 3. Assume that a vertex u of degree three or greater exists in $\langle O \rangle$. Let $\{x, y, z\} \subseteq N_{|O}(u)$. Since O has no neighbors in I, by Claim 2, we can find a path P to u from a leaf $v \in L$. If P contains the edge (u, v) then we can construct a new tree in which we add the vertices u, x, y and z. We lose v as a leaf in T but we gain x, y and z and thus $(G, r, k + 1)$ would be a YES-instance for ANNOTATED MAX LEAF SPANNING TREE, a contradiction. If P does not contain the edge (u, v) then without loss of generality assume that P goes through x before reaching u. If we extend T to include P then we can add two leaves to T, namely y and z, while only losing one leaf, v. Thus, $(G, r, k + 1)$ is a YES-instance for ANNOTATED MAX LEAF SPANNING TREE, contradicting our hypothesis.   □

Claim 4 *There are no cycles in $\langle O \rangle$.*

Proof of Claim 4. Assume there exists a cycle C in $\langle O \rangle$. Since O has no neighbors in I, by Claim 2, we can find a path P to a vertex $x \in C$ from a leaf v in L. Since x is part of a cycle, it has two neighbors in O, namely y and z not in P. If we extend T to include P, y and z then y and z can be added as

leaves, while only losing one leaf, v. Thus, $(G, r, k+1)$ is a YES-instance for ANNOTATED MAX LEAF SPANNING TREE, contradicting our hypothesis. □

Note that by Claims 3 and 4, the graph $\langle O \rangle$ induced by O is composed of disjoint paths as there are no cycles and no vertex has degree greater than two.

Claim 5 *Internal vertices of the disjoint paths of $\langle O \rangle$ have degree two in G, i.e., they are not adjacent to vertices in T.*

Proof of Claim 5. Assume there exists a vertex x in one of the disjoint paths and that x is adjacent to a leaf v in T. Assume x is an internal vertex to the path, with two neighbors $y, z \in O$. If we extend T to include x, y and z then we can add two leaves to T, namely y and z, while only losing one leaf, v. Thus, $(G, r, k + 1)$ is a YES-instance for ANNOTATED MAX LEAF SPANNING TREE, a contradiction.   □

Claim 6 *The maximum length of a path in $\langle O \rangle$ is two, i.e., each one of the disjoint paths in $\langle O \rangle$ has at most three vertices.*

Proof of Claim 6. Assume there exists a path in $\langle O \rangle$ with length three or greater. By Claim 5, the internal vertices in the path have degree two in G. By Reduction Rule 1 there are no bridges in G. Then, there are two adjacent vertices of degree two in G and the edge between them is not a bridge. This contradicts the hypothesis that (G, r, k) is reduced under Reduction Rule 2.□

Claim 7 *The size of O is at most $\frac{3}{4}k$.*

Proof of Claim 7. There are no paths of length greater than two by Claim 6, thus there are only three possible types of structures in O which do not

contradict the fact that (G, r, k) is a reduced instance. These three types must be paths of length zero, one or two (types A, B and C, respectively, as shown in Figure 3.6). We know there are exactly k leaves in T and the endpoints of the paths in O are leaves by Claim 2. Hence, the worst case is provided by paths whose endpoints have as few neighbors in T as possible.
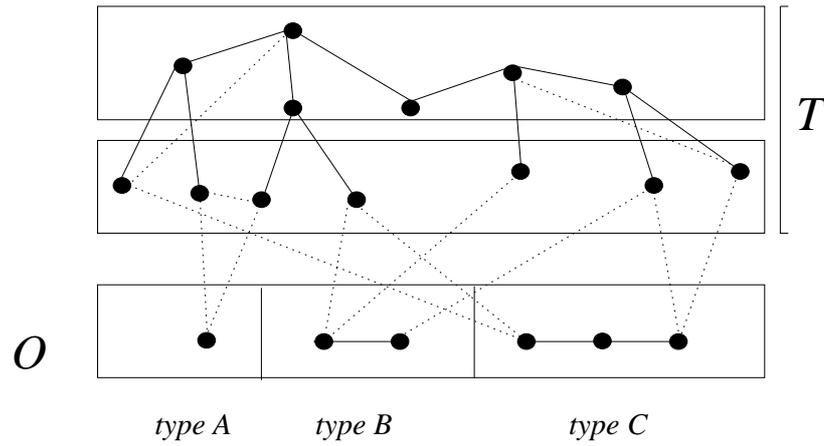
Figure 3.6: Example of the three types of paths in O.

- A path of type A must have at least two different neighbors in L as there are no vertices of degree one in O by Reduction Rule 3. Since we know that there are exactly k leaves in T, and we want to minimize the number of outsiders, the worst-case ratio of outsiders to leaves provided by this type of paths occurs when there is one outsider for every two leaves. The worst-case ratio is thus 2:1.

- A path of type B has at least three different neighbors in T. This is due to the fact that both endpoints of the path must have neighbors in T as there are no vertices of degree one in G by Reduction Rule 3, and if each endpoint had only one neighbor in G then we would have two adjacent vertices of degree two and the instance would not be reduced under Reduction Rules 1 and 2. The worst-case ratio provided by this type of path is 3:2.

- A path of type C has at least four different neighbors in T. This is due to

the fact that both endpoints of the path must have neighbors in T, there are no vertices of degree one in G by Reduction Rule 3, and if either endpoint had only one neighbor in G then we would have two adjacent vertices of degree two and the instance would not be reduced under Reduction Rules 1 and 2. The worst-case ratio provided by this type of path is 4:3.

The worst ratio of outsiders to leaves is provided by type C paths, requiring three outsiders for every four leaves in the worst case. This implies the $\frac{3}{4}k$ bound stated by the claim. $\square$

Claim 8 *If $|V| > 2$, then for every leaf v in L the parent of v in T is in I_B.*

Proof of Claim 8. If $|V| > 2$ then any leaf in L must have at least one neighbor v' in I such that $(v, v') \in E(T)$. Assume in contradiction that $v' \notin I_B$. Then the degree of v' in T is two and we can construct a tree T' with k leaves by removing the edge (v, v') from T. In this new tree v' is a leaf and thus (G, r, k) is a YES-instance for ANNOTATED MAX LEAF SPANNING TREE. However, $|T'| < |T|$, contradicting the first inductive priority. $\square$

Claim 9 *The size of I_B is at most $k - 2$.*

Proof of Claim 9. We prove this claim by induction on k.

- *(Base Case)* If T has only two leaves then T is a path and thus there are no branching vertices in T. The size of I_B is at most $k - 2$, as required.

- *(Inductive Hypothesis)* Assume it is true that any tree with $k - 1$ leaves has at most $k - 3$ branching vertices.

- *(Inductive Step)* Consider the tree T with k leaves. If $|V| = 2$ then any tree in G has at most $k = 2$ leaves and we are in the base case. Otherwise, if $|V| > 2$, by Claim 8 the parent w of any leaf v in T

must be a branching vertex. Let T' be the subtree obtained from T by removing v. Since w is a branching vertex and T' has $k-1$ leaves, by the inductive hypothesis T' has at most $k-3$ branching vertices. In T, two cases arise:

- If w is a branching vertex in T' then T has at most $k-3$ branching vertices. As $k-3 < k-2$, the number of branching vertices in T is at most $k-2$, as required in the inductive step.

- If w is not a branching vertex in T' then it becomes a branching vertex for T when we add v. Thus T has at most $k-3+1 = k-2$ branching vertices, completing the proof of Claim 9.

$\square$

So far, a bound has been found for the size of L, O and I_B. To complete the proof we must also bound the size of $I \setminus I_B$. We say that two vertices in I_B are consecutive if there is a path P in T between them containing only vertices in $I \setminus I_B$. Note that all vertices in this path are internal vertices with degree two in T. We say that an edge (u,v) is a non-tree edge if $(u,v) \notin T$.

Claim 10 *There are at most three vertices in $I \setminus I_B$ between every two consecutive vertices in I_B.*

Proof of Claim 10. Let P be the path in T between two consecutive branch vertices. Assume in contradiction that there are at least four non-branch vertices in P. Assume that $\{u, v, v', w\}$ are the first four vertices in this path. We assume that these four vertices are ordered from left to right, giving an orientation to the tree. Let r be the root of the tree T.

By Reduction Rule 1, there are no bridges in a reduced instance (G, r, k). In particular, the edge $(v, v') \in E$ cannot be a bridge in G. Hence, by Reduction

Rule 2, there exists at least one non-tree edge between v or v' and a vertex x in G. Without loss of generality assume that vertex is v'. We are going to analyze all possible cases for the position of vertex x.

- Vertex x cannot be in O as there are no non-tree edges between vertices in I and vertices in O by Claim 2. Thus x must be either in I or in L.

- Vertex x cannot be in the subtree to the left of v, T_v as otherwise a tree T' can be constructed by adding (x, v') to T' and removing (u, v) as shown in Figure 3.7.
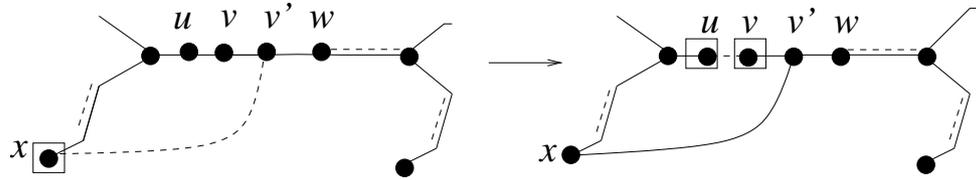
Figure 3.7: Transformation from T to T' when x is in T_v.

The resulting tree T' is a new witness tree. In the worst case x was a leaf and we have lost x as a leaf of the tree, but gained two leaves, namely u and v. Thus, $(G, r, k+1)$ is YES-instance for ANNOTATED MAX LEAF SPANNING TREE, contradicting the hypothesis of Lemma 3.3.

- Vertex x cannot be internal in $T_{v'}$, as otherwise a tree T' can be constructed by adding (x, v') to T' and removing (v', w). In T' vertex w is a leaf and thus $(G, r, k+1)$ is YES-instance for ANNOTATED MAX LEAF SPANNING TREE, contradicting the hypothesis of Lemma 3.3.

- Therefore the non-tree edge should be between v' and a leaf $x \in T_{v'}$. Let x' be the parent of x in T. Two cases arise depending on the position of r, the root of T:

  • The root r is in the subtree to the left of v, T_v. A tree T' can be constructed by adding (x, v') to T' and removing (x, x'). In T' vertex x

is still a leaf. However, the distance between the root r and x has decreased by at least two. All other distances from r to the rest of leaves in T remain equal, and thus we have reached a contradiction with the second inductive priority.

- The root r is in the subtree to the right of v', $T_{v'}$ (see Figure 3.8). By Reduction Rule 1, there are no bridges in a reduced instance (G, r, k). In particular, the edge $(u, v) \in E$ cannot be a bridge in G. Hence, by Reduction Rule 2, there exists at least one non-tree edge between u or v and a vertex y in G. Using a symmetrical argument to the one described above we can see that y cannot be O or in $T_{v'}$. Thus, there exists at least one non-tree edge between u or v and a vertex y in T_v. Let y' be the neighbor of y in a path P from y to u in T.
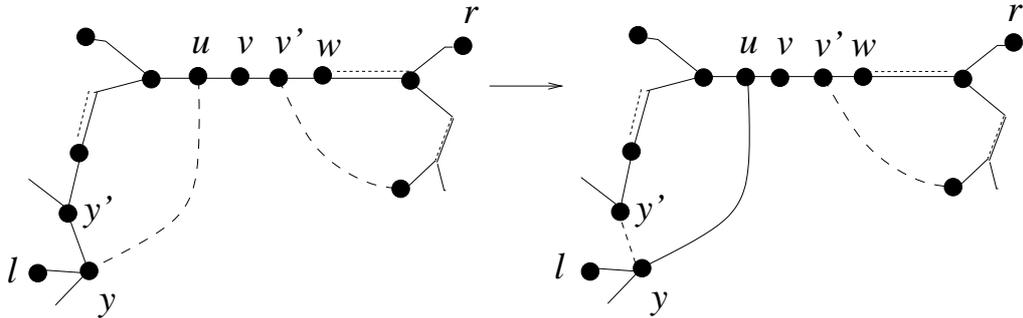
Figure 3.8: Transformation from T to T' when r is in $T_{v'}$.

A tree T' can be constructed by adding (v, y) or (u, y) to T' and removing (y, y'). Since r is in the subtree $T_{v'}$, all leaves in the subtree to the left of y', $T_{y'}$, are at the same distance from the root as they were in T. However, the distance between the r and the leaves in subtree to the right of y has decreased by at least two. This contradicts the second inductive priority.

$\square$

Claim 11 *The number of non-branching vertices in I is at most $3(k-3)$.*

Proof of Claim 11. By Claim 9 there are at most $k-2$ vertices in I_B. There are thus at most $k-3$ positions in T where the non-branching vertices between vertices in I_B could be placed. By Claim 10 there are at most three non-branching vertices between every two vertices in I_B. Thus, there are at most $3(k-3)$ non-branching vertices between any vertices in I_B. By Claim 8 no non-branching vertices can occur between vertices in I_B and vertices in L. Hence, the total number of non-branching vertices is at most $3(k-3)$. $\square$

The total number of vertices is thus $|V| = |L| + |I_B| + |I \smallsetminus I_B| + |O|$. Since T has k leaves by the hypothesis in the boundary lemma, $|L| = k$. By Claim 9, $|I_B| \leq k-2$ and by Claim 11, $|I \smallsetminus I_B| \leq 3(k-3)$. The number of outsiders, $|O|$, is at most $\frac{3}{4}k$ by Claim 7. Thus, $|V| \leq k + k - 2 + 3(k-3) + \frac{3}{4}k < 5.75k$ as stated by the boundary lemma. $\square$

Now we prove Lemma 3.2. This lemma states that if (G, r, k) is a reduced instance with a catalytic vertex $r \in V$, and $|V(G)| \geq 5.75k$, then (G, r, k) is a YES-instance for ANNOTATED MAX LEAF SPANNING TREE.

Proof of Lemma 3.2. (***Kernelization Lemma***) Assume in contradiction to the stated Lemma 3.2 that there exists a graph instance (G, r, k) of size $|V(G)| \geq 5.75k$ which has no k-leaf spanning tree.

Let $k' < k$ be the largest k' for which (G, r, k') is a YES-instance. By the boundary lemma we know that $|V(G)| < 5.75k' < 5.75k$. This contradicts the assumption. $\square$

ALGORITHM AND RUNNING TIME ANALYSIS

In the previous section we proved a kernel for ANNOTATED MAX LEAF SPANNING TREE of size $5.75k$ in the number of vertices. In Table 3.2 we provide an algorithm to solve MAX LEAF SPANNING TREE.

First, the algorithm applies Reduction Rules 1, 2 and 3. If, when these rules are no longer applicable, the instance does not contain vertices of degree one and two, then the following theorem, first conjectured by Linial and Sturtevant and proved in [KW91] by Kleitman and West, applies:

Theorem 3.1 *[KW91] Every connected graph $G = (V, E)$ with minimum degree at least three has a spanning tree with at least $\frac{1}{4}|V| + 2$ leaves. Moreover, such a spanning tree can be determined in polynomial time.*

Therefore, if there are no vertices of degree two in G then the graph has a spanning tree with $k' \geq \frac{1}{4}|V| + 2$ leaves. Thus, if in the instance (G, k) we have $|V| \geq 4k - 8$, then $k' \geq \frac{4k-8}{4} + 2 = k$ and we can answer YES.

If there are vertices of degree two, then the algorithm selects any of them, v. Since the instance is reduced under Reduction Rules 1, 2 and 3, both neighbors of v must have degree at least three. Furthermore, at least one of these neighbors must be internal to any spanning tree for G, as otherwise the tree would not be connected. Thus, one of them may be chosen as the catalytic vertex. Once the catalytic vertex is chosen we have a reduced instance of ANNOTATED MAX LEAF SPANNING TREE. This instance, as we showed in the previous section, has a kernel of size at most $5.75k$.

In the algorithm we call the subroutine in Table 3.1 to choose a set of k vertices in G and check whether they can be leaves of a tree T in G.

```
Step 1.  Try all ways of choosing a set S of k vertices
         in G.
Step 2.  For each such set S check if:
         2.1  V \ S is connected, and
         2.2  V \ S is a dominating set.
```

Table 3.1: Subroutine to solve MAX LEAF SPANNING TREE.

```
Step 1.  Apply Reduction Rules 1, 2 and 3 until they can
         no longer be applied.
Step 2.  If there are no vertices of degree two in G,
         then:
         2.1 If |V(G)| ≥ 4k − 8, then answer YES and
         halt.
         2.2 Else, apply the subroutine in Table 3.1 and
             go to Step 6.
Step 3.  Else, if there exists a vertex v in V of degree
         two, then let N(v) = {x, y}.
Step 4.  Make r = x and consider the corresponding in-
         stance of ANNOTATED MAX LEAF SPANNING TREE.
         4.1  If |V(G)| > 5.75k then answer YES and halt.
         4.2  Otherwise de-catalyze the instance and
              apply the subroutine in Table 3.1.  If
              the answer is YES for any set S in the
              subroutine, answer YES and halt.
Step 5.  Make r = y and consider the corresponding in-
         stance of ANNOTATED MAX LEAF SPANNING TREE.
         5.1  If |V(G)| > 5.75k then answer YES and halt.
         5.2  Otherwise de-catalyze the instance and
              apply the subroutine in Table 3.1.
Step 6.  If the answer is YES for any set S in the sub-
         routine, answer YES and halt.
Step 7.  Else answer NO and halt.
```

Table 3.2: FPT algorithm to solve MAX LEAF SPANNING TREE.

Lemma 3.4 *If G has n vertices, the subroutine in Table 3.1 can be solved in time $\mathcal{O}\big(n^2\binom{n}{k}\big)$.*

Proof of Lemma 3.4.

A graph G with n vertices has a spanning tree with k leaves if and only if G has a con-nected dominating set of size $n - k$. Thus if we can find a connected dominating set in $V \smallsetminus S$ then the graph G would certainly have a spanning tree with at least $|S|$ leaves, and thus (G, k) would be a YES-instance for MAX LEAF SPANNING TREE.

Checking if $V \smallsetminus S$ is connected can be done in linear time in $|V|$. Checking if $V \smallsetminus S$

dominates G requires to show if every element in S has at least one neighbor in S. Since each element in S has at most $n-1$ neighbors, dominance can be checked in quadratic time in n. There are $\binom{n}{k}$ ways of choosing k vertices in a set with n vertices, and thus the total running time for the subroutine is $\mathcal{O}(n^2\binom{n}{k})$. $\hfill\square$

Lemma 3.5 *The* Max Leaf Spanning Tree *problem can be solved in time* $\mathcal{O}^*(14.25^k)$.

Proof of Lemma 3.5.

Step 1: Reduction Rule 1 can be performed in time $\mathcal{O}(|E|)$ as it is concerned with identifying bridges in G and then performing a constant-time operation. Reduction Rule 3 can be performed only $|V|$ times. The action of making a clique out of the neighbors of the vertex can be carried out in quadratic time and thus the running time of this step is $\mathcal{O}(|V|^2)$.

Reduction Rule 2 can be performed in time $\mathcal{O}(|V|^2)$. Once the graph is reduced under Reduction Rule 1 there are no bridges in the graph so we only need to find adjacent vertices of degree two and then performing a constant-time operation.

Step 2: Checking if there are vertices of degree two can be done in linear time in $|V|$.
    Step 2.1: Checking if $|V| \geq 4k-8$ can be done in time linear in k.
    Step 2.2: As we showed in Lemma 3.4, the time to run the subroutine is $n^2\binom{n}{k}$. Since there are at most $4k-8$ vertices in the instance, the running time is bounded by $(4k)^2\binom{4k}{k}$. Using Lemma 1.1 in Chapter 1, this is at most $(4k)^2 \cdot 9.49^k$.

Step 3: This step can be performed in constant time.

Step 4: Making x the catalytic vertex can be done in constant time.
    Step 4.1: Checking if $|V| > 5.75k$ can be done in time linear in k.
    Step 4.2: As we showed in Lemma 3.4, the time to run the subroutine is $n^2\binom{n}{k}$. Since there are at most $5.75k$ vertices in the instance, the running time is bounded by $(5.75k)^2\binom{5.75k}{k}$. Using Lemma 1.1 in Chapter 1, this is at most $(5.75k)^2 \cdot 14.25^k$.

Step 5: The running time of Step 5 is exactly the same as the running time of Step 4, $\mathcal{O}\big((5.75k)^2 \cdot 14.25^k\big)$.

Step 6 and Step 7: Steps 6 and 7 can be carried out in constant time.

The total running time of the algorithm is thus $\mathcal{O}\big(|V|^2 + (5.75k)^2 \cdot 14.25^k\big) = \mathcal{O}^*(14.25^k)$ concluding the proof of Lemma 3.5. $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\square$

There exists an algorithm for MAX LEAF SPANNING TREE running in time $\mathcal{O}^*(9.49^k)$ [BBW03]. This chapter has been introduced in the thesis since it provides the proofs for the reduction rules sketched in [FM+00], giving the first example of an application to the *method of extremal structure* with the use of catalytic vertices.

NONBLOCKER

MOTIVATION

The problem of finding a dominating set of a given cardinality in a graph, the DOMI-NATING SET problem, is one of the oldest and most important combinatorial problems on graph theory. The problem is NP-complete even when restricted to planar graphs with maximum vertex degree three or to planar graphs that are regular of degree four [GJ79]. DOMINATING SET arises in numerous applications in countless fields, as testified by books dedicated to this topic [HHS98].

The DOMINATING SET problem is not only NP-complete, but its optimization version has been proved hard to approximate [AC+99, GJ79]. This problem is also intractable when viewed as a parameterized problem [DF99]. However, the problem in which we are asked to find a dominating set of size at most $|V| - k$, NONBLOCKER (also known as ENCLAVELESS SETS [HHS98]), is fixed-parameter tractable. McCartin gives a $\mathcal{O}^*(4^k)$ algorithm for this problem [McC03]. We use the following natural parameterization of NONBLOCKER:

> NONBLOCKER
>
> Instance: A graph $G = (V, E)$ and a positive integer k
>
> Parameter: k
>
> Question: Is there a subset $V' \subset V$ of size at least k where for each vertex
> $u \in V'$ there is a $v \in V \smallsetminus V'$ such that the edge $(u, v) \in E$?

The results presented in this chapter are joint work with F. Dehne, M. Fellows, H. Fernau and F. Rosamond [DF+03].

REDUCTION RULES

A subset of vertices V' such that every vertex in V' has a neighbor in $V \smallsetminus V'$ is denoted a *non-blocking set*. A vertex u in V' is said to be dominated by a vertex v in $V \smallsetminus V'$ if $(u, v) \in E$.

The reduction rules offer a specific feature in the case of NONBLOCKER: one of them makes use of a *catalytic vertex*, a vertex which we force to be outside the non-blocking set. Hence, here we define the corresponding annotated problem. Later in this section we will introduce a special de-catalyzation rule which will enable us to transform an instance of ANNOTATED NONBLOCKER into an instance of NONBLOCKER.

ANNOTATED NONBLOCKER

Instance: A graph $G = (V, E)$, a catalytic vertex c and a positive integer k

Parameter: k

Question: Is there a subset $V' \subset V$ of size k where for each vertex $u \in V'$ there is a $v \in V \smallsetminus V'$ such that the edge $(u, v) \in E$ and vertex c is in $V \smallsetminus V'$?

An instance of (G, k) NONBLOCKER with a catalytic vertex c is thus an instance of ANNOTATED NONBLOCKER and is denoted (G, c, k).

Reduction Rule 1 *Let (G, c, k) be an instance of* ANNOTATED NONBLOCKER. *Let $v \neq c$ be an isolated vertex in G. Let $G' = G \smallsetminus v$. Transform (G, c, k) into (G', c, k).*

Soundness Lemma 1 Reduction Rule 1 is sound.

Proof of Soundness Lemma 1. If v is an isolated vertex of G different from c, then any non-blocking set in G is also a non-blocking set in $G \smallsetminus v$ and vice versa, as v could never be in V'. $\qquad\qquad\qquad\qquad\square$

Reduction Rule 2 *Let (G, k, c) be an instance of* ANNOTATED NONBLOCKER. *Let $v \neq c$ be a vertex of degree one in G with $N(v) = u$. Transform (G, c, k) into $(G', c', k - 1)$, where:*

- *If $u \neq c$ then $G' = G_{[c \leftrightarrow u]} \smallsetminus v$, i.e., G' is the graph obtained by deleting v and merging u and c into a new catalytic vertex $c' = c \leftrightarrow u$.*
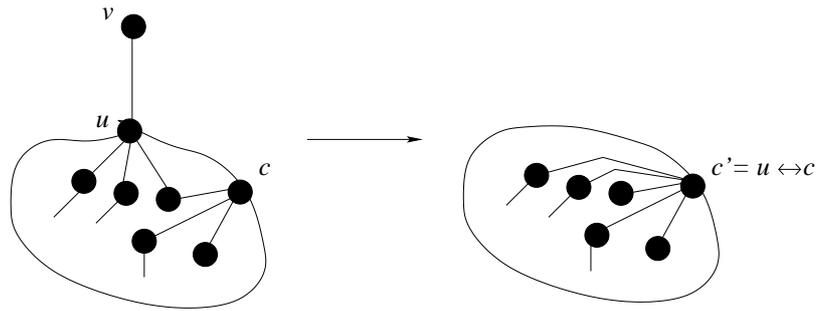
Figure 4.1: Reduction Rule 2 for ANNOTATED NONBLOCKER.

- *If $u = c$ then $G' = G \smallsetminus v$ and $c' = c$.*

Soundness Lemma 2 Reduction Rule 2 is sound.

Proof of Soundness Lemma 2.

($\Rightarrow$:) Let (G, c, k) be an instance of ANNOTATED NONBLOCKER. Let $V' \subset V(G)$ be a non-blocking set in G with $|V'| = k$. The vertex v is a vertex of degree one in G. Let u be the neighbor of v in G. Two cases arise:

1. If $v \in V'$ then it must have a neighbor in $V(G) \smallsetminus V'$ and thus $u \in V(G) \smallsetminus V'$. Deleting v will decrease the size of V' by one. If $u = c$, then $(G', c', k-1)$ is a YES-instance of ANNOTATED NONBLOCKER. If $u \neq c$, merging u and c will not affect the size of V' as both vertices are now in $V(G') \smallsetminus V'$. Thus, $(G', c', k-1)$ is a YES-instance of ANNOTATED NONBLOCKER.

2. If $v \in V(G) \smallsetminus V'$, then two cases arise:

   2.1. If u is also in $V(G) \smallsetminus V'$ then deleting v does not affect the size of V'. Note that this argument is valid whether $u = c$ or $u \neq c$.

   2.2. If $u \in V'$ then $u \neq c$. If we make $v \in V'$ and $u \in V(G) \smallsetminus V'$, the size of V' remains unchanged. Since u did not dominate any vertices in the graph, this change does not affect $N(u) \smallsetminus v$, and Case 1 now applies.

($\Leftarrow$:) Conversely, assume that $(G', c', k-1)$ is a YES-instance of ANNOTATED NONBLOCKER. Two cases arise:

1. If $u = c$, then we can always place v in V' and thus (G, c, k) is a YES-instance for ANNOTATED NONBLOCKER.

2. If $u \neq c$, getting from G' to G can be seen as (1) splitting the catalytic vertex c' into two vertices c and u, (2) taking c as the new catalytic vertex, and (3) attaching a pendant vertex v to u. As the vertex u is in $V(G) \smallsetminus V'$, v can always be placed in V', increasing the size of this set by one. Thus (G, c, k) is a YES-instance for ANNOTATED NONBLOCKER, concluding the proof of Lemma 2.

$\square$

Reduction Rule 3 *Let (G, c, k) be an instance of* ANNOTATED NONBLOCKER. *Let u and v be two adjacent vertices of degree 2, neither of them the catalytic vertex. Let*

$N(u) = \{v, w\}$ *and* $N(v) = \{u, w\}$. *Transform* (G, c, k) *into* $(G', c', k - 2)$ *where:*

- *If* $w \neq c$ *then let* $G' = G_{[c \leftrightarrow w]} \setminus \{u, v\}$, *i.e.,* G' *is the graph obtained by deleting* u *and* v *and merging* w *and* c *into a new vertex* $c' = c \leftrightarrow w$.
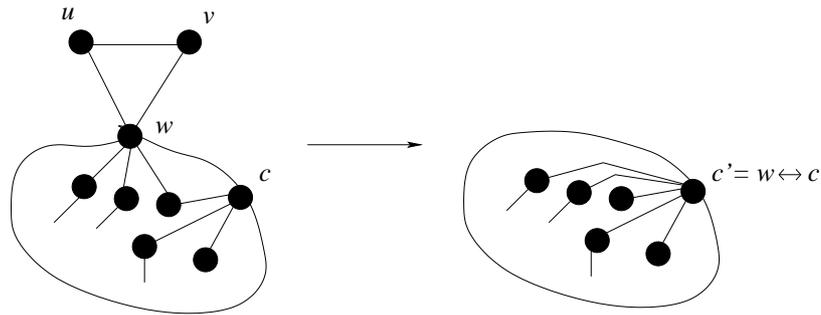
Figure 4.2: Reduction Rule 3 for ANNOTATED NONBLOCKER.

- *If* $w = c$ *then let* $G' = G \setminus \{u, v\}$ *and* $c' = c$.

Soundness Lemma 3   Reduction Rule 3 is sound.

Proof of Soundness Lemma 3.

($\Rightarrow$:) Assume that (G, c, k) is a YES-instance. Let $V' \subset V(G)$ be a non-blocking set in G with $|V'| = k$. Two cases arise:

1. If w is in $V(G) \setminus V'$. Then, u and v are dominated by w, and they can both be in V'. Two cases arise:

   1.1. If $c = w$, then, since at most two vertices from V' have been deleted, $(G', c', k - 2)$ is a YES-instance for ANNOTATED NON-BLOCKER.

   1.2. If $c \neq w$, then when merging w with the catalytic vertex we are merging two vertices in $V(G) \setminus V'$. All their former neighbors

will thus be dominated by c'. Since at most two vertices from V' have been deleted, $(G', c', k-2)$ is a YES-instance for ANNOTATED NONBLOCKER.

2. If w is in V', then it is not the catalytic vertex and it is not dominating any vertices in $N(w) \setminus \{u, v\}$. At most one vertex out of $\{u, v\}$ is in V'. Assume, without loss of generality, that it is u. If we place v in V' and w in $V(G') \setminus V'$ we obtain an equivalent solution where Case 1.2 applies. Thus, $(G', c', k-2)$ is a YES-instance for ANNOTATED NONBLOCKER.

($\Leftarrow$:) Conversely, assume that $(G', c', k-2)$ is a YES-instance of ANNOTATED NONBLOCKER. Let V'' be a non-blocking set in G' with $|V''| = k - 2$. Two cases arise:

1. If $w = c$, then we can always place u and v in V' and thus (G, c, k) is a YES-instance for ANNOTATED NONBLOCKER.

2. If $w \neq c$, getting from G' to G can be seen as (1) splitting the catalytic vertex c' into two vertices c and w, (2) taking c as new catalytic vertex, and (3) attaching a dangling triangle with vertices u and v to w. As the vertex w is in $V(G) \setminus V'$, u and v can always be placed in V' increasing the size of this set by two. Thus, (G, c, k) is a YES-instance for ANNOTATED NONBLOCKER, concluding the proof of Lemma 3.

$\square$

Reduction Rule 4 *Let (G, c, k) be an instance of* ANNOTATED NONBLOCKER. *Let u and v be two adjacent vertices of degree two, neither of them the catalytic vertex.*

Let $N(u) = \{u', v\}$ and $N(v) = \{v', u\}$ with $u' \neq v'$. Let $G' = G_{[u' \leftrightarrow v']} \setminus \{u, v\}$, i.e., G' is the graph obtained by deleting u and v and merging u' and v' into a new vertex $u' \leftrightarrow v'$. Transform (G, c, k) into $(G', c', k - 2)$ where:

- *If one of the two vertices u' or v' is the catalytic vertex, then $c' = u' \leftrightarrow v'$.*
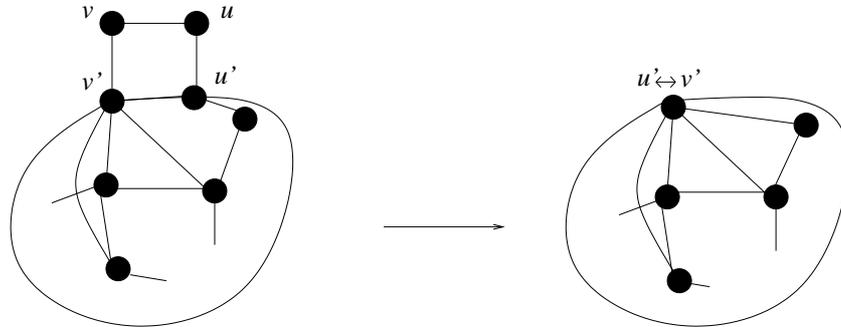
Figure 4.3: Reduction Rule 4 for ANNOTATED NONBLOCKER.

- *If neither of the two vertices u' or v' is the catalytic vertex, then $c' = c$.*

Soundness Lemma 4 Reduction Rule 4 is sound.

Proof of Soundness Lemma 4.

($\Rightarrow$:) Assume that (G, c, k) is a YES-instance. Let $V' \subset V(G)$ be a non-blocking set in G with $|V'| = k$. Several cases arise:

1. Both u and v are in V'. Then, u' and v' must be in $V(G) \smallsetminus V'$. After the reduction the merged element can be placed in $V(G') \smallsetminus V'$ and two vertices in V' have been deleted. Moreover, all former neighbors of u' and v' will be dominated by $u' \leftrightarrow v'$. Exactly two elements in the non-blocking set have been deleted and thus $(G', c, k-2)$ is a YES-instance for ANNOTATED NONBLOCKER. Since $u' \leftrightarrow v' \in V(G') \smallsetminus V'$ this argument is valid whether or not one of the two vertices u' or v' is the catalytic vertex.

2. Both u and v are in $V(G) \smallsetminus V'$. The maximum number of vertices in the set $\{u', v'\}$ which could possibly be in V' is two. By transforming

G into G' and making $u' \leftrightarrow v'$ be in $V(G') \smallsetminus V'$ (dominating all former neighbors of u' and v'), we decrease the size of V' by at most two. Thus, $(G', c', k - 2)$ is a YES-instance for ANNOTATED NONBLOCKER. Since $u' \leftrightarrow v' \in V \smallsetminus V'$ this argument is valid whether one of the two vertices u' or v' is the catalytic vertex or not.

3. One of the vertices in the set $\{u, v\}$ is in V' and the other is in $V(G) \smallsetminus V'$. Assume without loss of generality that $u \in V'$. The maximum number of vertices in the set $\{u', v'\}$ which could possibly be in V' is two. Two cases arise:

   3.1. If only one vertex out of $\{u', v'\}$ is in V' then by transforming G into G' and making $u' \leftrightarrow v'$ be in $V(G') \smallsetminus V'$ (dominating all former neighbors of u' and v'), we decrease the size of V' by at most two and $(G', c', k - 2)$ is a YES-instance for ANNOTATED NONBLOCKER. Since $u' \leftrightarrow v' \in V(G') \smallsetminus V'$ this argument is valid whether one of the two vertices u' or v' is the catalytic vertex or not.

   3.2. If both u' and v' are in V', then neither of them can be the catalytic vertex. Since u' is in V', then u' must have a neighbor u'' in $V(G) \smallsetminus V'$. By transforming G into G' and making $u' \leftrightarrow v'$ be in V', we are decreasing the size of V' by at most two. Note that $u' \leftrightarrow v'$ can be dominated by u''. Thus, $(G', c', k - 2)$ is a YES-instance for ANNOTATED NONBLOCKER.

($\Leftarrow$:) Conversely, assume $(G', c', k - 2)$ is a YES-instance of ANNOTATED NONBLOCKER. Let V'' be a non-blocking set in G' with $|V''| = k - 2$. Two different possibilities must be analyzed:

1. If the vertex $u' \leftrightarrow v' \in V(G') \smallsetminus V''$, then by placing u' and v' into $V(G) \smallsetminus V'$, where V' is a dominating set for G, and putting u and

v into V', we create a new instance with two more elements in the non-blocking set, and (G, c, k) is thus a YES-instance for ANNOTATED NONBLOCKER. Note that this argument is valid whether $u' \leftrightarrow v' = c'$ or $u' \leftrightarrow v' \neq c'$.

2. If the vertex $u' \leftrightarrow v' \in V''$, then $u' \leftrightarrow v' \neq c'$. There exists a vertex $w \in V(G') \smallsetminus V''$ which dominates $u' \leftrightarrow v'$. Assume, without loss of generality, that in G the neighbor of w is u'. If we put u' into V' (dominated by w), and v into $V \smallsetminus V'$ (vertices u and v' now in V' dominated by v), we create a new instance with two more vertices in V'. The instance (G, c, k) is thus a YES-instance for ANNOTATED NONBLOCKER.

□

Reduction Rule 5 *Let (G, k, c) be an instance of* ANNOTATED NONBLOCKER. *Let G' be the graph obtained by deleting all edges between the vertices in $N(c)$. Transform (G, c, k) into (G', c, k).*

Soundness Lemma 5 Reduction Rule 5 is sound.

Proof of Soundness Lemma 5.

$(\Rightarrow:)$ All vertices in the neighborhood of c are dominated by c and thus if (G, c, k) is a YES-instance of ANNOTATED NONBLOCKER we do not need the edges between them to make sure that these vertices are in V'.

$(\Leftarrow:)$ Conversely, if we have a YES-instance with a catalytic vertex and we add some edges to it, all vertices that were in the non-blocking set in G', will still be in a non-blocking set in G, dominated by the same vertices they were dominated by in G'. The result will always be a YES-instance for ANNOTATED NONBLOCKER.

□

We consider an instance of ANNOTATED NONBLOCKER *reduced* when Reduction Rules 1 to 5 have been exhaustively applied.

In order to transform ANNOTATED NONBLOCKER to NONBLOCKER we introduce a special de-catalyzation rule. We apply this rule *once* to a reduced instance of ANNOTATED NONBLOCKER. This reduction rule is designed so as to produce a graph G of minimum degree at least two such that no two adjacent vertices of degree two occur in G.

Proposition 4.1 *(De-catalyzation Rule) Let (G, c, k) be an instance of* ANNOTATED NON-BLOCKER. *Let Reduction Rules 1 to 5 be no longer applicable to (G, c, k). Let G' be the graph obtained from G by adding three new vertices u, v, and w and edges (c, u), (c, v), (c, w), (u, v), and (v, w).*

(G, c, k) is a YES-*instance for* ANNOTATED NONBLOCKER *if and only if $(G', k + 3)$ is a* YES-*instance for* NONBLOCKER.
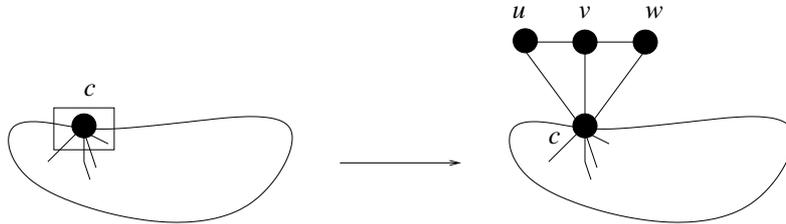
Figure 4.4: De-catalyzation Rule for ANNOTATED NONBLOCKER.

Proof of Proposition 4.1.

$(\Rightarrow:)$ Let (G, c, k) be a YES-instance of ANNOTATED NONBLOCKER. This means that there is a non-blocking set $V' \subset V(G)$ with $|V'| = k$, such that $c \notin V'$. In the instance $(G', k' = k + 3)$ constructed as described above, if we place c in $V(G') \smallsetminus V')$, the set $V' \cup \{u, v, w\}$ is a non-blocking set as u, v and w can be dominated by c. Thus, $(G', k+3)$ is a YES-instance for NONBLOCKER.

($\Leftarrow$:) Conversely, assume $(G', k + 3)$ is a YES-instance of NONBLOCKER, where G' is constructed from G (with catalytic vertex c) as explained above. Let V'' be a maximum non-blocking set with $|V''| \geq k + 3$. Two cases apply:

1. If $c \notin V''$, then by maximality $\{u, v, w\} \subset V''$. Therefore, in this case c could become a catalytic vertex in the instance (G, c, k) by making $c \notin V'$ in G. We can construct a solution for (G, c, k) where $V' = V'' \smallsetminus \{u, v, w\}$. There are three vertices fewer in V' than there were in V'' and $c \notin V'$. Thus (G, c, k) is a YES-instance of ANNOTATED NONBLOCKER.

2. If $c \in V''$, then at least one of $\{u, v, w\}$ is not in V''. This means that there is another solution V''' where c is not in the non-blocking set but all u, v and w are. Vertex c did not dominate vertices in $N(c) \smallsetminus \{u, v, w\}$ so this change does not affect the rest of the graph. The size of V''' is not smaller than $|V''|$, and c is in $V(G') \smallsetminus V'''$. Vertex c can be considered the catalytic vertex for (G, c, k), Case 1 applies, and thus (G, c, k) is a YES-instance of ANNOTATED NONBLOCKER.

$\square$

In the algorithm we propose for NONBLOCKER we start by choosing any catalytic vertex, then exhaustively applying to this annotated instance Reduction Rules 1 to 5 and, finally, de-catalyzing. Thus, we define each *reduced instance* of NONBLOCKER to be a reduced instance of ANNOTATED NONBLOCKER de-catalyzed using Lemma 4.1

Now, we state the main combinatorial property used to prove the bound on the kernel size.

Proposition 4.2 *Any reduced instance of* NONBLOCKER *has minimum degree two. Moreover, no two vertices of degree two are adjacent.*

Proof of Proposition 4.2. The following list of properties holds for an instance (G, k) of ANNOTATED NONBLOCKER where Reduction Rules 1 to 5 can no longer be applied:

1. If (G, c, k) contains an isolated vertex, it must be the catalytic vertex.

2. If (G, c, k) contains a vertex of degree one, it must be the catalytic vertex.

3. If (G, c, k) contains two consecutive vertices of degree two, one of them must be the catalytic vertex.

This can be seen as follows:

- If G contains an isolated vertex that is not catalytic, then Reduction Rule 1 applies.

- If G contains a vertex of degree one that is not catalytic, then Reduction Rule 2 applies.

- If G contains two adjacent vertices u, v of degree two, where neither of them is the catalytic vertex, then Reduction Rule 3 or 4 applies.

When we transform an instance of ANNOTATED NONBLOCKER into an instance of NON-BLOCKER, the de-catalyzation rule destroys these three possible contradictions to the hypotheses by increasing the degree of the former catalytic vertex by three. Since the vertices introduced have degree at least two, and there are no two adjacent vertices of degree two, we do not introduce any new violations. □

Observe that the de-catalyzation rule is only used once. This is important, since it is a transformation which actually enlarges the instance.

KERNELIZATION AND BOUNDARY LEMMAS

The following result by McCuaig and Shepherd will help us bound the number of vertices in a graph instance.

Theorem 4.1 *[McS89] If a connected graph G has minimum degree two and is not one of seven exceptional graphs, then the size of its minimum dominating set is at most $\frac{2}{5}|V|$.*

The exceptional seven graphs mentioned above are the ones represented in Figure 4.5.
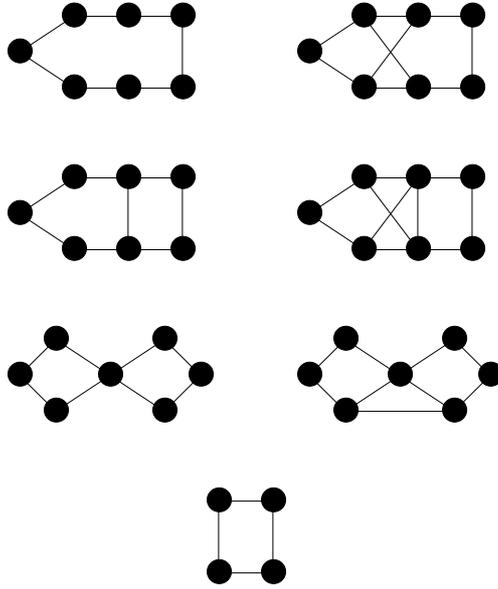
Figure 4.5: Collection of exceptional graphs.

Lemma 4.1 *(**Boundary Lemma**) Let (G, k) be reduced instance of* NONBLOCKER. *Let* (G, k) *be a* YES-*instance and* $(G, k + 1)$ *be a* NO-*instance for* NONBLOCKER. *Then, the number of vertices in* G *is at most* $\frac{5}{3}k$.

Proof of Lemma 4.1. By Proposition 4.2 any reduced instance (G, k) of NONBLOCKER has minimum degree two and no adjacent vertices of degree two.

If (G, k) is a YES-instance and $(G, k + 1)$ is a NO-instance of NONBLOCKER, that means that k is the size of the maximum non-blocking set in G.

By Theorem 4.1 the minimum dominating set of any graph with minimum degree two is at most $\frac{2}{5}|V|$, unless the graph is one of the seven exceptional graphs. Hence if k' is the size of the minimum dominating set for G, $k' \leq \frac{2}{5}|V|$. Since NONBLOCKER is the parametric dual of the DOMINATING SET problem, we know that $k' = |V| - k$ where k is the size of the maximum non-blocking set in G. Thus, if $|V| - k \leq \frac{2}{5}|V|$ then $|V| \leq \frac{5}{3}k$.

Note that all seven exceptional graphs in Figure 4.5 have two adjacent vertices of degree two and, by Proposition 4.2, would never constitute reduced instances of NONBLOCKER. □

Lemma 4.2 (***Kernelization Lemma***) *If a graph instance* (G, k) *is reduced and has more than* $\frac{5}{3}k$ *vertices, then it is a* YES-*instance of* NONBLOCKER.

Proof of Lemma 4.2. Assume in contradiction to Lemma 4.2 that there exists a graph instance (G, k) of size $|V(G)| > \frac{5}{3}k$ such that there does not exist a non-blocking set of size k for G.

Let $k' < k$ be the largest k' for which G is a YES-instance. By Lemma 4.1 we know that $|V(G)| \leq \frac{5}{3}k' < \frac{5}{3}k$. This contradicts the assumption.   □

ALGORITHM AND RUNNING TIME ANALYSIS

In the previous section we proved a kernel for the problem of size $\frac{5}{3}k$ in the number of vertices. In Table 4.1 we provide an algorithm to solve NONBLOCKER.

Lemma 4.3 NONBLOCKER *can be solved in time* $\mathcal{O}^*(3.07^k)$.

Proof of Lemma 4.3. We analyze each step in the algorithm separately.

Step 0: There are only $|V|$ possible ways of choosing the catalytic vertex c in G when we are annotating the instance.

Step 1: Reduction Rule 1 can be performed in linear time in the size of V as there are at most $|V|$ isolated vertices in G. There are at most $|V| - 1$ vertices of degree one in G and merging two vertices can be carried out in linear time in the size of V, hence Reduction Rule 2 can be performed in quadratic time on the size of V. There are at most $|V|$ vertices of degree two in G and thus Reduction Rules 4 and 3 can also be performed

```
Step 0.  Annotate the instance (G, k) in all possible ways
         by choosing a catalytic vertex c.
Step 1.  For each annotated instance (G, c, k) apply Re-
         duction Rules 1 to 5 until all of them can no
         longer be applied.
Step 2.  De-catalyze the instance as shown in Lemma 4.1.
Step 3.  If |V(G)| > 5/3 k then answer YES and halt.
Step 4.  Construct all possible subsets S of size k in
         the problem kernel.
Step 5.  For each S constructed in Step 4 check if all
         vertices in V(S) have at least one neighbor in
         V(G) ∖ V(S).
Step 6.  If the answer is YES in any set S in Step 5,
         answer YES and halt.
Step 7.  Else answer NO and halt.
```

Table 4.1: FPT algorithm to solve NONBLOCKER.

in quadratic time as it is also linear to merge vertices. There are $\mathcal{O}(|V|^2)$ edges in G and thus Reduction Rule 5 can be carried out in quadratic time on the size of V.

Step 2: The reduction rule shown in Lemma 4.1 can be performed in constant time as it is only carried out once and it involves the addition of a fixed number of vertices.

Step 3: This step can be performed in linear time in the size of V.

Step 4: Once we have left at most $5/3k$ vertices, we calculate all possible subsets of size k in the kernel. According to Lemma 1.1 in Chapter 1, there are at most 3.07^k such subsets in G.

Step 5: This step can be performed in quadratic time on the size of the kernel as we must check for each vertex whether at least one of its neighbors is in the non-blocking set, giving a running time of $\mathcal{O}(k^2)$.

Steps 6 and 7: This steps can be carried out in constant time.

The running time of the whole process is thus bounded by $\mathcal{O}(|V|(|V|^2 + k^2 \cdot 3.07^k)) = \mathcal{O}^*(3.07^k)$, concluding the proof of Lemma 4.3. $\qquad\square$

STAR PACKING

MOTIVATION

The problem of MAXIMUM H-MATCHING, also known as MAXIMUM H-PACKING, is of practical interest in the areas of scheduling [BM02], wireless sensor tracking [BK+01], and wiring-board design and code optimization [HK78] among others.

The problem is defined as follows: Let $G = (V, E)$ be a graph and $H = (V_H, E_H)$ be a fixed graph. A H-packing for G is a collection of disjoint subgraphs of G, each isomorphic to H.

In an optimization sense, the problem that we want to solve is to find the maximum number of vertex disjoint copies of H that are subgraphs of G. The decision version of the problem is NP-complete when the graph H has at least three vertices in some connected component [HK78]. Note that in the case where H is the complete graph on two nodes, H-packing is the very well studied (and polynomial time solvable) problem MAXIMUM MATCHING. MAXIMUM H-PACKING has been thoroughly studied in terms of approximation. The problem has been proved to be APX-complete [Kann94] and approximable within $|V_H|/2 + \varepsilon$ for any $\varepsilon > 0$ [HS89]. Several restrictions have also been considered (planar graphs, unit disk graphs etc.) in terms of the complexity of their approximation algorithms. For a review of these results we refer the reader to [AC+99].

We explore the parameterized complexity of the MAXIMUM H-MATCHING problem for the case when H is a star, a subgraph isomorphic to $K_{1,s}$. In the next two sections we

will analyze two variants of the problem using two different approaches to the *method of extremal structure*. Firstly, we will use the approach to the kernelization and boundary lemmas in the same way as in all previous chapters to solve the general case for stars with $s \geq 3$. Then, we will use an algorithmic approach to the *method of extremal structure* for the case when $s = 2$.

The results in this chapter are joint work with C. Sloper and have been reported in [PS04].

5.1   PACKING OF GENERAL STARS

We use the following natural parameterization of s-STAR PACKING:

> s-STAR PACKING
>
> Instance: A graph $G = (V, E)$ and a positive integer k
>
> Parameter: k
>
> Question: Are there k vertex-disjoint instances of $K_{1,s}$ in G?

REDUCTION RULES

Reduction Rule 1 *Let G be a graph such that there exists $v \in V$ with $deg(v) \geq k(s + 1) - 1$. Let $G' = G \smallsetminus v$. Transform (G, k) into $(G', k - 1)$.*

Soundness Lemma 1   Reduction Rule 1 is sound.

> ***Proof of Soundness Lemma 1.***
>
> ($\Rightarrow$:) If G has a $K_{1,s}$-packing of size k then G' has a $K_{1,s}$-packing of size $k - 1$ as v cannot be in two different stars.
>
> ($\Leftarrow$:) If G' has a $K_{1,s}$-packing of size $k - 1$ we can create a $K_{1,s}$-packing of size k by adding v. The $k - 1$ stars already packed cannot use more than $(s + 1)(k - 1)$ of v's neighbors, leaving s vertices for v to form a new star

of size s. The proof of Lemma 1 is completed since $(s+1)(k-1) + s = k(s+1) - 1$. $\square$

Reduction Rule 2 *Let $G = (V, E)$ be a graph such that there exists $u, v \in V$ with $(u, v) \in E$. Let $deg(u) \leq deg(v) < s$. Let G' be G after deleting the edge (u, v). Transform (G, k) into (G', k).*

Soundness Lemma 2   Reduction Rule 2 is sound.

Proof of Soundness Lemma 2.

$(\Rightarrow:)$ If $deg(u) \leq deg(v) < s$ then u and v will never be the centers of any star of size s. Thus, the edge (u, v) will never be part of a packing of $K_{1,s}$.

$(\Leftarrow:)$ If G' has a $K_{1,s}$-packing P of size k, then P is a $K_{1,s}$-packing of size k for G. The edges added are not part of the packing. $\square$

KERNELIZATION AND BOUNDARY LEMMAS

We consider a graph to be *reduced* if Reduction Rules 1 and 2 can no longer be applied.

Lemma 5.1 *(**Kernelization Lemma**) If (G, k) is reduced and has more than $k(s^3 + ks^2 + ks + 1)$ vertices, then (G, k) is a* YES-*instance for s-*STAR PACKING.

To prove the kernelization lemma we need to prove first Lemma 5.2, the boundary lemma.

Lemma 5.2 *(**Boundary Lemma**) Let (G, k) be reduced under the set of reduction rules above. Let (G, k) be a* YES-*instance and $(G, k+1)$ be a* NO-*instance for s-*STAR PACK-ING. *Then the number of vertices of G is at most $k(s^3 + ks^2 + ks - s^2 - s + 1)$.*

Proof of Lemma 5.2. (By minimum counterexample) Assume (G, k) is a YES-instance and $(G, k+1)$ is a NO-instance for s-STAR PACKING. Assume $|V(G)| > k(s^3 + ks^2 + ks - s^2 - s + 1)$.

Consider, as a witness to the fact that (G, k) is a YES-instance of s-STAR PACKING, the following partition of V:

- $V(W)$, the vertices in a $K_{1,s}$-packing W of size k,

- $O = V \smallsetminus V(W)$, the outsiders.
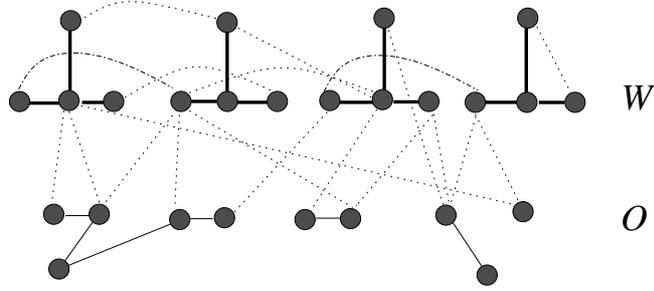
Figure 5.1: Example of witness structure with $s = 3$ and $k = 4$.

Let O_i be the set of vertices in O that have degree i in $\langle O \rangle$, the subgraph induced by O. The number of vertices in W is by definition $k \cdot (s+1)$. We will now prove a series of claims that bound the number of vertices in O.

Structural Claims

Claim 1 $\forall i \geq s, O_i = \varnothing$.

> *Proof of Claim 1.* This can be readily seen by contradiction. Otherwise we could add one more star of size s to W and $(G, k+1)$ would be a YES-instance for s-STAR PACKING. □

Claim 2 *Any s-star $S \in W$ has at most $s(s-1) + k(s+1) - 4$ neighbors in O.*

Proof of Claim 2. Assume that there is a reduced graph with a star S that has more than $s(s-1) + k(s+1) - 4$ neighbors in O. Let $u \in V(S)$ be the center of S. Two cases arise:

- Vertex u has neighbors in O. In this case all other vertices in S must have at most $(s-1)$ different neighbors in O as, if one vertex $v \in S$ had s different neighbors in O, we could form two new stars: one centered in u with all vertices in S except v and one of u's neighbors in O; and one centered in v with its s neighbors in O, contradicting the assumption that $(G, k+1)$ is a NO-instance for s-STAR PACKING. By Reduction Rule 1, u can have at most $k(s+1) - s - 2$ neighbors in O and, hence, the total number of neighbors of S is at most $k(s+1) - s - 2 + s(s-1)$. This is at most $s(s-1) + k(s+1) - 4$ when $s \geq 2$, contradicting the hypothesis.

- Vertex u has no neighbors in O. Two cases arise:

  - No other vertex in S has at least s neighbors in O. Then S has at most $s(s-1)$ different neighbors in O.

  - At least one vertex in S has at least s neighbors in O. Without loss of generality assume this vertex is v. Then each other vertex in the star has at most $s - 2$ different neighbors in $O \smallsetminus N(v)$, as otherwise we could form two stars. By Reduction Rule 1, v can have at most $k(s+1) - 3$ neighbors in O and, hence, the total number of neighbors of S is at most $k(s+1) - 3 + (s-2)(s-1) = k(s+1) + s^2 - 3s - 1$. This is at most $s(s-1) + k(s+1) - 4$ when $s \geq 2$, contradicting the hypothesis.

In the worst case, each star S in W has at most $k(s+1) + s(s-1) - 4$ neighbors in O. $\qquad\square$

The following claim follows directly from Claim 2, noting that, by the assumptions in Lemma 5.2, there are k stars in W.

Claim 3 W *has at most* $k \cdot (k(s+1) + s(s-1) - 4)$ *neighbors in* O.

Let now $O' = V \setminus (W \setminus N(W))$, i.e., O' is the set of vertices of O which do not have neighbors in W. By Claim 3, thus, $|O \setminus O'| \leq k \cdot (k(s+1) + s(s-1) - 4)$.

Claim 4 *The set* O' *is an independent set in* G.

> *Proof of Claim 4.* Assume in contradiction that there exist two vertices $u, v \in O'$ such that $(u, v) \in E$. By Claim 1 we know that both u and v have degree less than s in $\langle O \rangle$. Hence we have an edge between vertices of degree less than s, contradicting the assumption that the instance is reduced under Reduction Rule 2. □

Claim 4 ensures that all vertices in O' have an edge to one or more vertices in O. By Claim 1 we know that each of the vertices in O have at most $s - 1$ such neighbors and thus by Claim 3 we know that the total size of O' is at most $(s - 1) \cdot |O \setminus O'|$.

In total, G has size $|V(G)| = |W| + |O| = |W| + |O \setminus O'| + |O'| = |W| + |O \setminus O'| + (s-1)|O \setminus O'| = |W| + s|O \setminus O'| \leq k(s+1) + s \cdot k \cdot (k(s+1) + s(s-1) - 4) = k(s^3 + ks^2 + ks - s^2 - 3s + 1)$ contradicting the assumption that the graph had more than $k(s^3 + ks^2 + ks - s^2 - 3s + 1)$ vertices. This concludes the proof of the boundary lemma. □

Now we prove Lemma 5.1. This lemma states that any reduced instance with at least $k(s^3 + ks^2 + ks - s^2 - 3s + 1)$ vertices, is a YES-instance for s-STAR PACKING.

Proof of Lemma 5.1. Assume in contradiction to Lemma 5.1 that there exists a graph instance (G, k) of size $|V(G)| > k(s^3 + ks^2 + ks - s^2 - 3s + 1)$ which has no $K_{1,s}$-packing of size k.

Let $k' < k$ be the largest k' for which G is a YES-instance. By the boundary Lemma 5.2 we know that $|V(G)| \leq k'(s^3 + k's^2 + k's - s^2 - 3s + 1) < k(s^3 + ks^2 + ks - s^2 - 3s + 1)$. This contradicts the assumption.   □

ALGORITHM AND RUNNING TIME ANALYSIS

In the previous section we proved a kernel of quadratic size in the number of vertices. In Table 5.1 we provide an algorithm to solve s-STAR PACKING.

```
Step 1.  Apply Reduction Rules 1 and 2 to (G,k) until the
         instance is reduced.
Step 2.  If |V(G)| ≥ k(s³ + ks² + ks - s² - 3s + 1) then answer YES
         and halt.
Step 3.  Try all ways of choosing a set C of k vertices
         in G to generate all possible centers for the
         stars.
Step 4.  Find if there are k stars of size s for each
         subset of centers.
Step 5.  If the answer is YES for any C, answer YES and
         halt.
Step 6.  Else answer NO and halt.
```

Table 5.1: FPT algorithm to solve s-STAR PACKING.

Lemma 5.3 s-STAR PACKING *can be solved in time* $\mathcal{O}^*(4^{k \log k})$.

Proof of Lemma 5.3. We will analyze each step of the algorithm separately.

Step 1. Reduction Rules 1 and 2 can be carried out in quadratic time on the size of $V(G)$, as they compare the degrees of pairs of vertices in G.

Step 2. This step can be performed in linear time in the number of vertices.

Step 3. After the instance has been reduced, there are $\mathcal{O}(k^2)$ vertices. Thus, the number of subsets C of vertices of size k we can have is at most $\binom{ck^2}{k} \leq \left(ck^2\right)^k = 2^{\log(ck^2)k} = 2^{k \log(k\sqrt{c})^2} = \mathcal{O}(4^{k \log k})$.

Step 4. For each such set C, checking if there is a star of size s for all centers can be carried out in time $\mathcal{O}(k^6)$ [EK72] as, once the centers have been chosen, finding the stars can be reduced to a network flow problem. A network flow problem considers a graph with a set of sources S and sinks T such that each edge has an assigned capacity. The aim of the problem is to find the maximum flow that can be routed from S to T while respecting the given edge capacities.

We reduce our problem to a network flow problem by constructing an auxiliary graph G' with the property that G has k stars with the selected centers in S if and only if G' has a maximum flow of value ks:
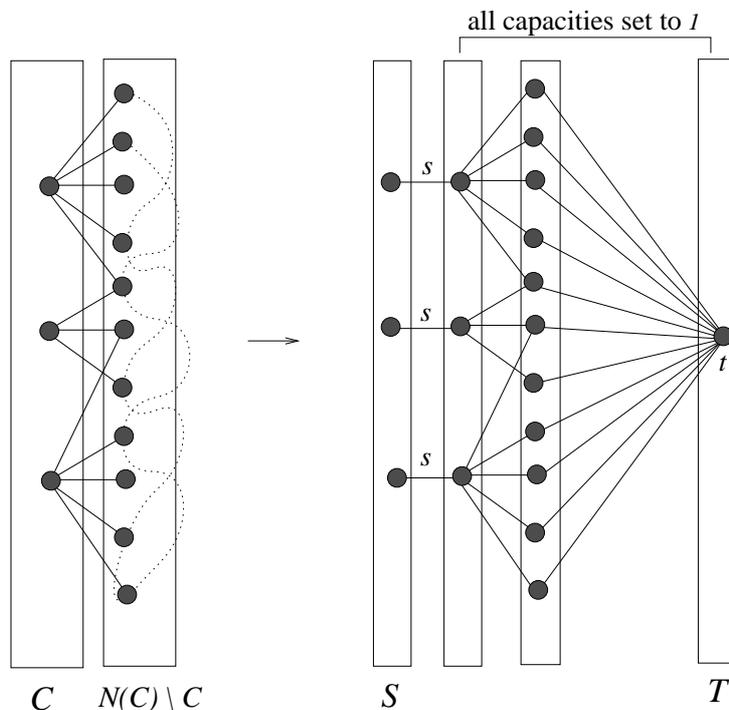

Figure 5.2: Transformation into a network flow problem.

- For each vertex v in $N[C]$, create a vertex v' in G'.
- For each edge from C to $N(C) \smallsetminus C$ in G, add an edge to G' with capacity 1.
- For each vertex u in C, create a vertex u in G'. These vertices are the sources S in G'.

Note that in G' we have now two copies of all vertices in C.

- Add an edge (u', v') with capacity s between the two copies u' and v' of vertices in C.
- Create a new vertex t in G', the sink.
- For each vertex in $N(C) \smallsetminus C$, add an edge to t with capacity 1.

Step 5 and Step 6. These two steps are solvable in constant time.

The running time of the whole process is thus bounded by $\mathcal{O}(|V|^2 + 4^{k \log k} \cdot k^6) = \mathcal{O}^*(4^{k \log k})$, concluding the proof of Lemma 5.3. $\qquad\square$

5.2 LINEAR PACKING OF SMALL STARS

In this section we consider the packing of stars of size two. Instead of using the kernelization and boundary lemmas to find theoretical bounds for the size of the problem kernel, we use an algorithmic approach to the *method of extremal structure*. The algorithmic version of the *method of extremal structure* operates by establishing a polynomial time algorithm aimed at creating extra structure to reduce the size of the instance. This algorithm starts by finding a maximal set which will act as the witness structure used in the boundary lemma in the previous chapters. Once this maximal set is computed we can proceed to transform the instance (I, k) using the reduction rules. The next phase is to prove that if $|I| > f(k)$ for some function f, then the preprocessing algorithm will either find a solution for $\mathcal{P}$ or it will reduce I.

We use the following natural parameterization of 2-STAR PACKING:

2-STAR PACKING

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

Parameter: k

Question: Are there k disjoint instances of $K_{1,2}$ in G?

REDUCTION RULES

In this section we modify the crown decomposition described in Chapter 1. Modifications

of the crown decomposition to suit specific problems are generally known as *modeled crown* decompositions. The two modeled crown decompositions we present in this section lead to two reduction rules for the 2-STAR PACKING problem.

The first modeled crown is called a *double crown decomposition*. In a double crown decomposition each vertex in H has two vertices from C matched to it (as opposed to only one).

Definition 5.1 *A* double crown decomposition (C, H, R) *in a graph* $G = (V, E)$ *is a partition of the vertices of the graph into three sets* H, C, *and* R *with the following properties:*

1. H (the head) *is a separator in* G *such that there are no edges in* G *between vertices belonging to* C *and vertices belonging to* R.

2. $C = C_u \cup C_m \cup C_{m'}$ (the crown) *is an independent set in* G, *partitioned into three disjoint subsets.*

3. $|C_m| = |H|$, $|C_{m'}| = |H|$ *and there is a perfect matching between* C_m *and* H, *and a perfect matching between* $C_{m'}$ *and* H.
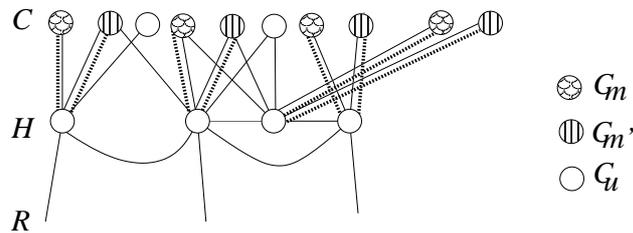
Figure 5.3: Example of a double crown.

The dashed lines in Figure 5.3 indicate how each vertex in H is matched to two vertices in C.

Another variation of the crown is the *fat crown* decomposition where instead of independent vertices in C we have independent complete graphs on two vertices, K_2's, as shown in Figure 5.4.

Definition 5.2 *A* fat crown decomposition (C, H, R) *in a graph* $G = (V, E)$ *is a partitioning of the vertices of the graph into three sets* H, C *and* R *with the following properties:*

1. H *(the head) is a separator in* G *such that there are no edges in* G *between vertices belonging to* C *and vertices belonging to* R.

2. $\langle C \rangle$ *is a forest where each component is isomorphic to* K_2.[1]

3. $|C| \geq |H|$, *and there is a perfect matching* M *between the vertices in* H *and a subset of cardinality* $|H|$ *in* C.

4. *Only one endpoint of each* K_2 *in* C *can be in* M.[2]
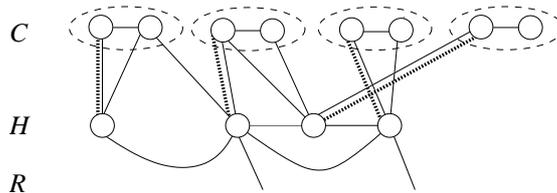
Figure 5.4: Example of fat crown.

The dashed lines in Figure 5.4 indicate the matching between H and C. The dashed ellipses in C show to which K_2 the vertex in H is matched.

Reduction Rule 3 *Let* (G, k) *be a graph instance that admits a double crown decomposition* (C, H, R). *Let* G' *be* G *after deleting* H *and* C. *Transform* (G, k) *into* $(G', k - |H|)$.

---

[1]We say that $\langle C \rangle$ is a collection of *independent* K_2's.

[2]Without loss of generality then, we can say that each vertex in H is matched to a K_2 in C.

Soundness Lemma 3 Reduction Rule 3 is sound.

Proof of Soundness Lemma 3.

($\Leftarrow$:) If G' has a $K_{1,2}$-packing of size $k - |H|$ then G has a $K_{1,2}$-packing of size k as $\langle H \cup C \rangle$ has a $|H|$-packing which does not use any vertices in G'. This holds since for all $v \in H$, the vertex v and the two vertices in C matched to v form a $K_{1,2}$.

($\Rightarrow$:) Let (G, k) be a YES-instance of 2-STAR PACKING such that G admits a double crown decomposition (C^*, H^*, R^*). Assume for contradiction that $(G', k - |H^*|)$ is a NO-instance for 2-STAR PACKING. This implies that $\langle H^* \cup C^* \rangle$ contains more than $|H^*|$ independent $K_{1,2}$'s. Since H^* is a separator, and C^* is an independent set in the graph, every $K_{1,2}$ in G that has vertices in $H^* \cup C^*$ must have at least one vertex in H^*. Thus, we can have at most $|H^*|$ independent $K_{1,2}$'s, a contradiction. $\square$

Reduction Rule 4 *Let (G, k) be a graph instance that admits a fat crown decomposition (C, H, R). Let G' be G after deleting H and C. Transform (G, k) into $(G', k - |H|)$.*

Soundness Lemma 4 Reduction Rule 4 is sound.

Proof of Soundness Lemma 4.

($\Leftarrow$:) If G' has a $K_{1,2}$-packing of size $k - |H|$ then G has a $K_{1,2}$-packing of size k as $\langle H \cup C \rangle$ has a $|H|$-packing. This is true since for all $v \in H$, the vertex v and the two vertices in the K_2 to which v is matched in C form a $K_{1,2}$.

($\Rightarrow$:) Let (G, k) be a YES-instance of 2-STAR PACKING such that G admits a fat crown decomposition (C^*, H^*, R^*). Assume for contradiction that $(G', k - |H^*|)$ is a NO-instance for 2-STAR PACKING. This implies that

$\langle H^* \cup C^* \rangle$ contains more than $|H^*|$ disjoint $K_{1,2}$'s. Since H^* is a separator, and C^* is a set of independent K_2's in the graph, every $K_{1,2}$ in G that has vertices in $H^* \cup C^*$ must have at least one vertex from H^*. Thus we can have at most $|H^*|$ independent $K_{1,2}$'s, a contradiction. □

The following two corollaries to Lemma 1.3 help to determine when we can find fat crown decompositions and double crown decompositions in a graph.

Corollary 5.1 *Any graph G with a collection J of independent K_2's such that $|N(J)| < |J|$, has a non-trivial fat crown decomposition (C, H, R), where $H \subseteq N(J)$. This decomposition can be found in time $\mathcal{O}(|V| + |E|)$, given J.*

Proof of Corollary 5.1. The corollary follows from Reduction Rule 4. If each K_2 is replaced with a single vertex, then, by Lemma has a crown, this graph has a crown. We can reintroduce the K_2's to obtain a fat crown. □

Corollary 5.2 *Any graph G with an independent set I, where $|I| > 2|N(I)|$, has a non-trivial double crown decomposition (C, H, R), where $H \subseteq N(I)$. This decomposition can be found in time $\mathcal{O}(|V| + |E|)$, given I.*

Proof of Corollary 5.2.

Let G be a graph with an independent set $I \subseteq V(G)$ such that $2|N(I)| < |I|$. We construct a graph $G' = G$, and add for every vertex $v \in N(I)$ a copy v' such that $N(v') = N(v)$. By Lemma 1.3, G' has a crown decomposition (C', H', R') such that $H' \subseteq N_{G'}(I)$. We use this crown to construct a double crown (H, C, R) in G.

First, observe that $v \in H'$ if and only if $v' \in H'$. Assume in contradiction that there exists a crown decomposition such that $v \in H'$ but $v' \notin H'$. The vertex v must be matched to some vertex u in C'. Since $N(v) = N(v')$, the vertex v' cannot be in C' as this would

contradict the fact that C' is an independent set in G. Also, v' is not in R' as this would contradict H' being a separator. Thus v' must be in H', contradicting the assumption.

Based on this observation, the result follows as H' consists of pairs of vertices, a vertex and its copy. Each pair v and v' in H' is matched to two vertices u_1 and u_2. In G, let v be in H and let it be matched to both u_1 and u_2. Doing this for every pair in H', we form a double crown in G. $\qquad\square$

KERNELIZATION AND BOUNDARY LEMMAS

In Table 5.2, we describe a polynomial time preprocessing algorithm to apply to an instance (G, k) of 2-STAR PACKING.

```
Step 1.  Compute with a greedy algorithm a maximal K_{1,2}-
         packing W.  Let O = V \ V(W).
Step 2.  Let I be the isolated vertices in ⟨O⟩.  While
         |I| > 2|N(I)| in G, by Corollary 5.2 there is a
         double crown in G and we can reduce (G,k) using
         Reduction Rule 3.
Step 3.  Let X be the components in ⟨O⟩ isomorphic to
         K_2.  While |X| ≥ |N(X)| in G, by Corollary 5.1
         there is a fat crown in G and we can reduce
         (G,k) using Reduction Rule 4.
```

Table 5.2: Preprocessing algorithm for 2-STAR PACKING.

Lemma 5.4 *(Algorithmic Boundary Lemma) If $|V(G)| > 15k$ then the preprocessing algorithm in Table 5.2 will either find a $K_{1,2}$-packing of size k or it will reduce (G, k).*

Proof of Lemma 5.4. Assume in contradiction to Lemma 5.4 that $|V(G)| > 15k$, but that the algorithm produced neither a $K_{1,2}$-packing of size k nor a reduction of (G, k).

After the preprocessing algorithm has been run, $V(G)$ is partitioned as follows:

- $V(W)$, the vertices in the maximal packing W for G.

- $O = V \smallsetminus V(W)$, the vertices not in present in W. We will call these vertices *outsiders*. Let O_i be the vertices in O that have degree i in $\langle O \rangle$, the subgraph induced by O.
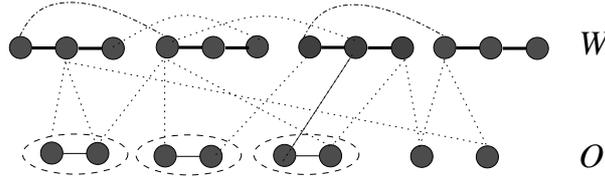
Figure 5.5: Example of structure after preprocessing algorithm has been run.

Note that the assumption implies that the maximal packing W is of size $|W| < 3k$, since if the packing was larger we would already have a YES-instance for 2-STAR PACKING.

Structural Claims

Claim 5 $\forall i \geq 2, O_i = \varnothing$.

Proof of Claim 5. Claim 5 is true as otherwise we could find an instance of $K_{1,2}$ in $\langle O \rangle$ and thus W would not be maximal. □

Claim 6 *The size of O_1 is at most $6k$.*

Proof of Claim 6. Assume in contradiction that $|O_1| > 6k$. This implies that the size of X, the set of K_2's in O, is greater than $3k$, but then $|X| > |W|$. By Corollary 5.1 G has a fat crown and should have been reduced in Step 2 of the algorithm by Reduction Rule 4, contradicting the hypothesis that no reduction can be performed. □

Claim 7 *The size of O_0 is at most $6k$.*

Proof of Claim 7. Assume in contradiction that $|O_0| > 6k$. Then $|O_0|$ is greater than $2|W|$ and by Corollary 5.2 G has a double crown and should have been reduced in Step 3 of the algorithm by Reduction Rule 3, contradicting the hypothesis that no reduction can be performed. $\square$

Thus the total size $|V(G)| = |W|+|O_0|+|O_1|+|O_2|+\sum_{i>2}|O_i| \leq 3k+6k+6k+0 = 15k$. This contradicts the assumption that $|V(G)| > 15k$.

$\square$

Lemma 5.5 *(Algorithmic Kernelization Lemma) Any instance (G,k) can be reduced to a problem kernel of size at most $15k$.*

Proof of Lemma 5.5. This follows from Lemma 5.4, as we can run the preprocessing algorithm until it fails to reduce (G,k). $\square$

ALGORITHM AND RUNNING TIME ANALYSIS

In the previous section we obtained a kernel for the problem of size $15k$ on the number of vertices. In Table 5.3 we provide an algorithm to solve 2-STAR PACKING.

```
Step 1.  Apply the preprocessing algorithm in Table 5.2
         to the instance (G, k).
Step 2.  If |V(G)|> 15k answer YES and halt.
Step 3.  Else find all possible subsets C of size k in
         (G, k).  For each C check if its elements are
         centers to k disjoint stars of size two.
Step 4.  If the answer is YES for one subset C in Step
         2, answer YES and halt.
Step 5.  Else answer NO and halt.
```

Table 5.3: FPT algorithm to solve 2-STAR PACKING.

Lemma 5.6 2-STAR PACKING *can be solved in time $\mathcal{O}^*(2^{5.301k})$.*

Proof of Lemma 5.6.

Step 1: The preprocessing algorithm finds a maximal packing in the reduced instance. This can be done in time $\mathcal{O}(|E|)$. Once the packing is found, Reduction Rules 3 and 4 are triggered. The running time of this reduction rule is linear in the number of the edges of the graph [CFJ04].

Step 2: The running time of Step 2 is linear in the size of V as we have to check all vertices in the graph.

Step 3: We find the center vertices of the $K_{1,2}$'s in a *brute force* manner. There are $\binom{15k}{k}$ ways to do this. By Stirling's formula this is bounded by $2^{5.301k}$. With k center vertices already selected the problem reduces to a problem on bipartite graphs where the question is if the left hand side each can have two neighbors assigned to it. This can easily be transformed to MAXIMUM BIPARTITE MATCHING by making two copies of each vertex on the left hand side. MAXIMUM BIPARTITE MATCHING can be solved in time $\mathcal{O}(\sqrt{|V|}|E|)$ [HK73]. We have $15k + k$ vertices, and thus $\mathcal{O}(k^2)$ edges. We can solve each try in time $\mathcal{O}(k^{2.5})$. The total running time of this step is $\mathcal{O}(2^{5.301k} \cdot k^{2.5})$.

Steps 4 and 5: These two steps can be carried out in constant time.

The running time of the whole process is thus bounded by $\mathcal{O}(|E| + |V| + k + 2^{5.301k} \cdot k^{2.5})$. This can be written as $\mathcal{O}^*(2^{5.301k})$, concluding the proof of Lemma 5.6. $\qquad\square$

EDGE-DISJOINT TRIANGLE PACKING

MOTIVATION

Let $G = (V, E)$ be a simple graph. A *triangle T* in G is any induced subgraph isomorphic to K_3. A graph $G = (V, E)$ is said to have a k-packing of triangles if there exist k triangles in G. The packing is called vertex-disjoint if no two of the k triangles share a vertex and is called edge-disjoint no two of the k triangles share an edge. We look at the following parameterization of the edge-disjoint case.

> EDGE-DISJOINT TRIANGLE PACKING (ETP)
>
> Instance: A graph $G = (V, E)$, a positive integer k
>
> Parameter: k
>
> Question: Are there at least k edge-disjoint triangles in G?

In a classical complexity setting, the decision problem ETP is NP-hard for general graphs [Hol81] and has been shown to be NP-hard for planar graphs, even if the maximum degree is five. It has applications in computational biology [BP96]. Regarding approximability, ETP is known to be APX-complete [Kann94]. A general result of [HS89] leads to a polynomial time $(3/2 + \epsilon)$ approximation algorithm for any $\epsilon > 0$ for this problem. If G is planar there is a polynomial time approximation scheme [Bak94] for the vertex-disjoint case, by methods that can be extended to yield a PTAS for ETP. Caprara and Rizzi prove that the problem remains APX-hard even for planar graphs with maximum degree five [CR01]. The problem of finding k vertex-disjoint triangles has been studied within the

framework of parameterized complexity [FH+04].

The results in this chapter are joint work with L. Mathieson and P. Shaw and have been reported in [MPS04]. The algorithm in Table 6.2 corrects the one appearing in [MPS04].

REDUCTION RULES

Reduction Rule 1 *Let u and v be two adjacent vertices of degree two in V. Let u and v be adjacent to the same vertex $w \in V$. Let G' be G after deleting u and v. Transform (G, k) into $(G', k - 1)$.*

Soundness Lemma 1 Reduction Rule 1 is sound.

Proof of Soundness Lemma 1.

$(\Rightarrow:)$ The vertices u, v and w form a triangle in G. The vertices u and v cannot be part of more than one triangle in G. By deleting u and v we have reduced the number of triangles in G by at most one and thus $(G', k - 1)$ is a YES-instance of EDGE-DISJOINT TRIANGLE PACKING.

$(\Leftarrow:)$ Adding the vertices u and v to G' creates a new triangle without destroying any possibilities as w can be used in more than one triangle. Hence (G, k) is a YES-instance of EDGE-DISJOINT TRIANGLE PACKING. □

Reduction Rule 2 *If a vertex v is not part of any triangle then transform (G, k) into $(G' = G \setminus \{v\}, k)$.*

Reduction Rule 3 *If an edge e is not part of any triangle then transform (G, k) into $(G' = G \setminus \{e\}, k)$.*

The soundness of the previous two reduction rules is trivial, as v and e will never participate in any triangle packing.

An edge $(u, v) \in E$ is *spanned* by a vertex w if (u, w) and (v, w) are in E. A set V' of vertices *spans* a set E' of edges if for every $e \in E'$ there exists a vertex in V' spanning e. We will use $S(V')$ to denote the set of all edges spanned by vertices in V' and $V(E')$ to denote the set of endpoints of edges in E'. Now we are going to present another example of modeled crown such as those shown in Chapter 5.

Definition 6.1 *A* fat-head crown decomposition *of a graph $G = (V, E)$ is a triplet (C, H, X) such that C and X form a partition of $V(G)$, $H \subseteq E(G)$ and the following properties hold:*

1. *C is an independent set of vertices.*

2. *H is the set of edges spanned by C, i.e., $H = S(C)$.*

3. *$V(H)$, the endpoints of the edges in H, forms a separator such that there are no edges from C to $X \setminus V(H)$.*

4. *there exists a function f mapping each edge $(h_1, h_2) \in H$ to a unique pair of distinct edges (h_1, u), (h_2, u) where $u \in C$.*
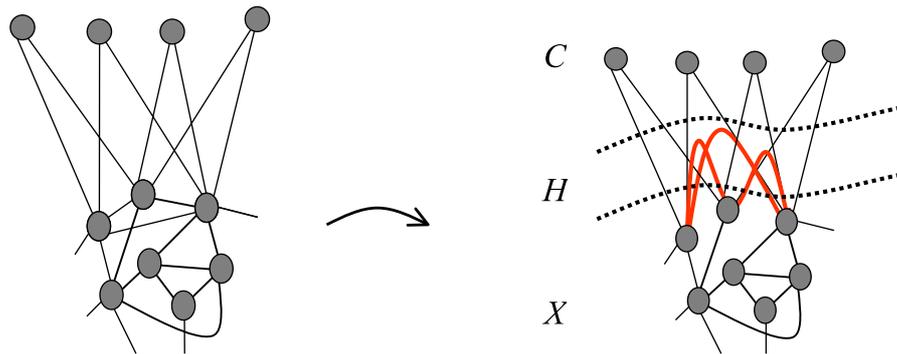
Figure 6.1: Example of fat-head crown.

Reduction Rule 4 *Let (G, k) be a graph instance that admits a fat-head crown decomposition (C, H, X). Let G' be G after deleting H and C. Transform (G, k) into $(G', k - |H|)$.*
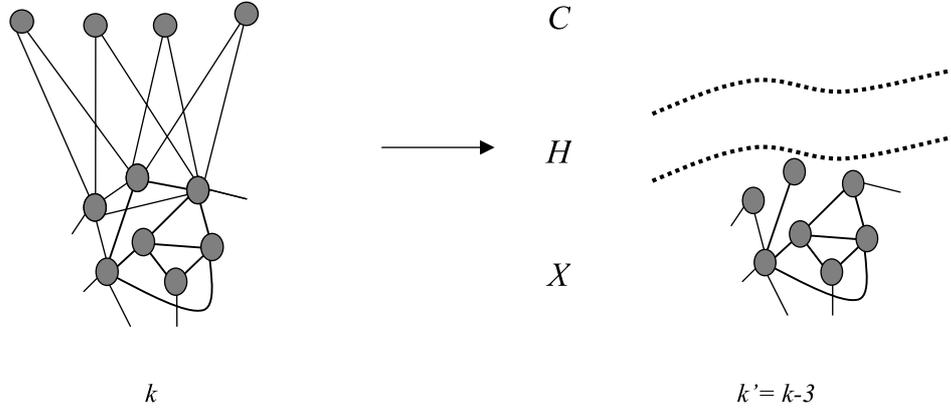
Figure 6.2: Reduction Rule 4 for EDGE-DISJOINT TRIANGLE PACKING.

Soundness Lemma 4 Reduction Rule 4 is sound.

Proof of Soundness Lemma 4.

$(\Rightarrow:)$ Let (G, k) be a YES-instance for EDGE-DISJOINT TRIANGLE PACK-ING admitting a fat-head crown decomposition. Consider how a k-packing P of triangles in G is affected by the deletion of H and C. Since C is an independent set in G and the degree of the vertices in C is at least two, any triangle we remove from the instance must include an edge of H. Hence at most $|H|$ triangles of P are removed by the transformation of G into G'. Thus G' admits a packing with at least $k' = k - |H|$ triangles.

$(\Leftarrow:)$ Assume that $(G', k - |H|)$ is a YES-instance of EDGE-DISJOINT TRI-ANGLE PACKING. We have a set of $|H|$ non-edges[1] between vertices in the graph G'. We add $|H|$ edges and for each one of these edges at least one vertex in C spanning that edge. These new vertices are independent and consequently each edge in H will contribute with exactly one triangle to the packing of G. We are adding $|H|$ triangles and (G, k) is a YES-instance. $\square$

---

[1]A non-edge is a pair of vertices u, v in G such that $(u, v) \notin E$.

The following corollary to Lemma 1.3 is used to determine when there exists a fat-head crown decomposition in a graph.

Corollary 6.1 *Any graph G with an independent set I, where $|I| > |S(I)|$, has a fat-head crown decomposition (C, H, X), where $H = S(I)$. This decomposition can be found in linear time in the size of G, given I.*

Proof of Corollary 6.1. Let G be graph with an independent set I such that $|I| > |S(I)|$. We construct a graph model $G' = (V', E')$ such that G admits a fat-head crown decomposition if and only if G' has a crown decomposition.

The new graph $G' = (V', E')$ is constructed from $G = (V, E)$ by making a copy of G and:

- For each edge $e_i \in S(I)$ add vertex v_i to V' and edges to the endpoints of e_i.

- For every u_j in I spanning the edge e_i, add (u_j, v_i) to E'.

- Remove from E' the edges from the endpoints of e_i to all u_j.
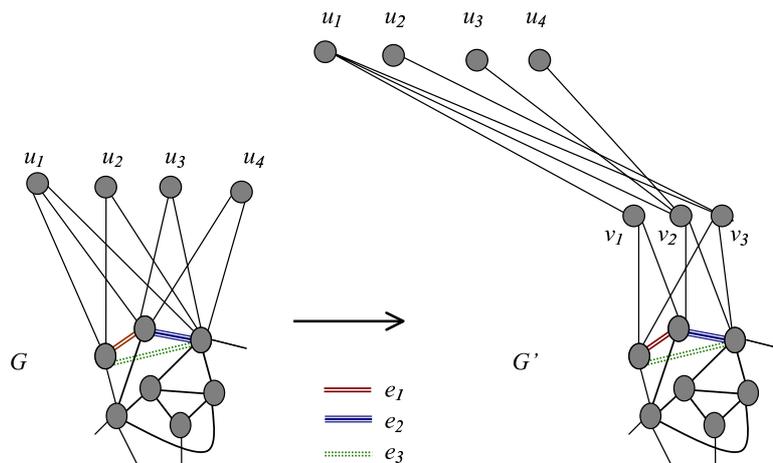
Figure 6.3: Example of transformation from G into G'.

($\Rightarrow$:) Let I' be the set of vertices of G' corresponding to I in G. By our construction it is clear that if $|I| > |S(I)|$ in G then $|I'| > |N(I')|$ in G' and, by Lemma 1.3 in Chapter 1, there exists a crown in G'.

($\Leftarrow$:) If G' has a crown decomposition (C', H', X'), then we can find a fat-head crown (C, H, X) in G where:

- $C = C'$ is an independent set by definition.

- H is the set of edges represented by the vertices in H' excluding its endpoints. That is, every vertex in $v \in H'$ corresponds to an edge in $(v_1, v_2) \in H$.

- By the definition of crown decomposition, there exists a matching between C' and H' so that each vertex v in H' can be univocally assigned a vertex u in C'. By the nature of the construction of G', each vertex in $v \in G'$ corresponds to an edge $(v_1, v_2) \in H$. The mapping f we are looking for between the elements of H and the vertices in C is thus the one assigning to each edge (v_1, v_2) the edges $(u, v_1), (u, v_2)$.

- X is the rest of the graph.

Thus, if $|I| > |S(I)|$ there exists a fat-head crown decomposition. $\qquad \square$

KERNELIZATION AND BOUNDARY LEMMAS

In Table 6.1, we describe an FPT preprocessing algorithm which we apply to an instance (G, k) *reduced* under Reduction Rules 1, 2 and 3.

Lemma 6.1 *(Algorithmic Boundary Lemma) If $|V(G)| > 4k$ then the preprocessing algorithm in Table 6.1 will either find an edge-disjoint triangle packing of size k or it will reduce (G, k).*

Step 1. Compute with a greedy algorithm a maxi-
         mal edge-disjoint triangle packing P. Let
         $O = V \smallsetminus V(P)$.
         (We will argue in Lemma 6.1 that the maxi-
         mality of P implies that O is an indepen-
         dent set in G).
Step 2. Find all possible subsets F of vertices in
         $V(P)$.
Step 3. For each of the subsets find all vertices
         in $N(F) \cap O = O'$ and calculate $S(O')$.
Step 4. If $|O'| > |S(O')|$ by Corollary 6.1 there is a
         fat-head crown that can be reduced using
         Reduction Rule 4.

Table 6.1: Preprocessing algorithm for EDGE-DISJOINT TRIANGLE PACKING.

Proof of Lemma 6.1. Assume in contradiction that $|V(G)| > 4k$, but that the algorithm produced neither a k edge-disjoint triangle packing nor a reduction of (G, k).

After the preprocessing algorithm has been run, $V(G)$ is partitioned as follows:

- $V(P)$, the vertices of the maximal edge-disjoint triangle packing P for G.

- $O \subset V \smallsetminus V(P)$, the vertices not present in P.
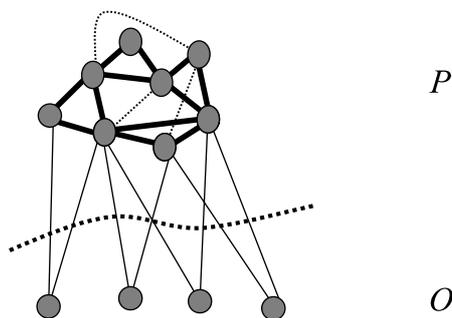
Figure 6.4: Example of structure after preprocessing algorithm has been run.

We will refer to the edges not participating in the packing as *ghost edges*. Note that P

contains at most $3k$ vertices, since if the packing was larger we would already have a YES-instance for EDGE-DISJOINT TRIANGLE PACKING.

Structural Claims

Claim 1 *There are no triangles in $\langle O \rangle$.*

Proof of Claim 1. If there existed a triangle in $\langle O \rangle$, then we could increase the size of the packing by one, contradicting our assumption that P is maximal. □

Claim 2 *The set O is an independent set in G.*

Proof of Claim 2. Assume that there exists an edge (u_1, u_2) for some $u_1, u_2 \in O$. Then by Reduction Rule 3, it must be part of some triangle, and by Claim 1, the triangle is not in O. So there exists a vertex $v \in P$ that forms this triangle. Then we have a triangle that is not in the packing, so we could increase the packing by one, contradicting our assumption that P is maximal. □

Claim 3 *For all $u \in O$, u does not span any ghost edges in P.*

Proof of Claim 3. Assume there was such a vertex. Then we would have a triangle that is not included in the packing, and we could increase the packing by one, contradicting the assumption that P is maximal. □

Claim 4 *For every triangle $\widehat{(u, v, w)} \in P$, if one edge is spanned by $x \in O$, then there is no $y \in O, y \neq x$ such that y spans another edge in $\widehat{(u, v, w)}$.*

Proof of Claim 4. Assume that we had such an arrangement. Then, without loss of generality, assume that x spans the edge (u, v) and y spans the edge (v, w). We can replace $\widehat{(u, v, w)} \in P$ with two triangles $\widehat{(u, v, x)}$ and $\widehat{(v, w, y)}$, and thus increase the size of the packing by one, contradicting the assumption that P is maximal. $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\square$

Note that if a vertex $x \in O$ spans more than one edge in a triangle $\widehat{(u, v, w)} \in P$, then it spans all three edges. Then by the previous claim, if a vertex $x \in O$ spans more than one edge in a triangle $\widehat{(u, v, w)} \in P$, then it is the only vertex that spans any edge in $\widehat{(u, v, w)}$.

From now on we consider in $\langle P \rangle$ the following:

- T_0: triangles in $\langle P \rangle$ with no vertices in O spanning any of their edges.

- T_1: triangles in $\langle P \rangle$ which have exactly one edge spanned by vertices in O.

- T_A: triangles in $\langle P \rangle$ which have all edges spanned by some unique vertex in O.

Note that $|T_0| + |T_1| + |T_A| < k$ as otherwise we would already have a k-packing of edge-disjoint triangles. Consequently, note that $|T_A| < k - |T_1| - |T_0|$.

We say that a vertex spans a triangle if it spans any edge in the triangle. We define O_0 to be the set of vertices in O spanning triangles of type T_0, O_A the set of vertices in O spanning triangles of type T_A and O_1 the set of vertices in $O \smallsetminus O_A$ spanning triangles of type T_1. Note that the vertices in O_1 are those vertices in O spanning only triangles of type T_1.

Claim 5 *The set $O_0 = \varnothing$ (by definition of T_0).*

Claim 6 *The size of O_A, $|O_A| < k - |T_1| - |T_0|$.*
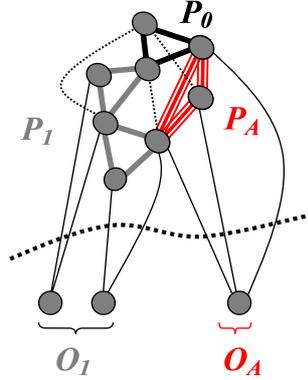
Figure 6.5: Example of partition of the sets O and P

Proof of Claim 6. Each T_A-triangle is spanned by a unique vertex in O. Also note that a vertex in O may span all edges of two triangles of type T_A. Then, $|O_A| \leq |T_A| < k - |T_1| - |T_0|$. □

Claim 7 *The size of O_1, $|O_1| \leq |T_1|$.*

Proof of Claim 7. By definition of T_1, only one edge in each triangle of type T_1 is spanned by vertices in O. Assume that $|O_1| > |T_1|$, then by Corollary 6.1, there is a fat-head crown in G, as O_1 is an independent set (by Claim 2) and the edges spanned by O_1 in T_1 constitute $S(O_1)$. This fat-head crown should have been reduced in Step 2 of the algorithm, contradicting the hypothesis that (G, k) can no further be reduced by Reduction Rule 4. □

Claim 8 *The total size of O, $|O| < k$.*

Proof of Claim 8. There are no vertices in O_0 by Claim 5, and consequently $|O| = |O_A \cup O_1|$. The size of O is thus, $|O_1 \cup O_A| \leq |O_1| + |O_A|$. By Claim 6 this is at most $|T_1| + |O_A|$, and $|T_1| + |O_A| < |T_1| + (k - |T_1| - |T_0|)$ by Claim 7. Hence, $|O| < k - |T_0| \leq k$. □

Claim 9 *The total size of V, $|V| \leq 3k + k = 4k$.*

Proof of Claim 9. As noted before, the size of $V(P)$ is at most $3k$, and the size of O is at most k by Claim 8. By definition $|V| = |V(P)| + |O|$ and the inequality follows. □

Thus from Claim 9, we have that the total size of $|V|$ is at most $4k$ contradicting our initial assumptions and thus we have the proof of the lemma. □

Lemma 6.2 *(**Algorithmic Kernelization Lemma**) Any instance (G, k) can be reduced to a problem kernel of size at most $4k$.*

Proof of Lemma 6.2. This follows from Lemma 6.1, as we can run the preprocessing algorithm until it fails to reduce G. □

ALGORITHM AND RUNNING TIME ANALYSIS

In the previous section we obtained a kernel for the problem of size $4k$ on the number of vertices. In Table 6.2 we provide an algorithm to solve EDGE-DISJOINT TRIANGLE PACKING.

```
Step 1.  Apply Reduction Rules 1,2 and 3 to (G,k) until
         the instance is reduced.
Step 2.  Apply the preprocessing algorithm in Table 6.1
         to the reduced instance (G,k) of Step 1.
Step 3.  If |V| > 4k answer YES and halt.
Step 4.  Apply subroutine in Table 6.3.
Step 5.  If the answer is YES in subroutine in Table
         6.3, answer YES and halt.
Step 6.  Else answer NO and halt.
```

Table 6.2: FPT algorithm to solve EDGE-DISJOINT TRIANGLE PACKING.

Lemma 6.3 EDGE-DISJOINT TRIANGLE PACKING *can be solved in time*
$\mathcal{O}^*\big(2^{3k(\log 3.2k)}\big)$.

Proof of Lemma 6.3. We will analyze each step of the algorithm separately.

Step 1: Reduction Rule 1 can be performed in quadratic time on $|V|$. Reduction Rules 2 and 3 have a worst-case running time of $\mathcal{O}(|V|^2 \cdot \binom{|V|}{2}) = \mathcal{O}(|V|^4)$.

Step 2: The preprocessing algorithm finds a maximal packing P in the reduced instance. This can be done in time $\mathcal{O}(|E|)$. As the packing has less than $3k$ vertices, calculating all subsets F in $V(P)$ can be done in time bounded by 2^{3k}. Finding the set $O' = N(F) \cap O$ and the set of edges spanned by O' can be done in quadratic time in the size of $V(G)$. Reduction Rule 4 may be triggered at this stage for every subset F. The running time of this reduction rule is linear in the number of edges in the graph [CFJ04].

The total time of the preprocessing algorithm is in $\mathcal{O}(|E| + 2^{3k}(|V|^2 + |E|))$.

Step 3: This step can be carried out in linear time in the number of vertices in the graph.

Step 4: We explore all ways of choosing k edge-disjoint triangles in the reduced instance. We do this by calling the following recursive process `findTriangles` on (G, k):

```
findTriangles(G, k);
if k = 0 return YES;
for each edge (u, v) ∈ E(G);
        for each w ∈ V(G), w ≠ u, v;
                if (u, v, w) is a triangle in G
                        do G' = (V, E ∖ {(u, v), (v, w), (w, u)});
                                return findTriangles(G', k − 1);
return NO.
```

Table 6.3: Subroutine to solve EDGE-DISJOINT TRIANGLE PACKING.

Line 2 of the algorithm can be executed in constant time. As there are at most $4k$ vertices and $8k^2 - 2k$ edges in G, line 5 of the algorithm is performed at most $(8k^2 - 2k)(4k - 2) <$

$32k^3$ times in each recursive call of `findTriangles`. Checking if three vertices form a triangle and then deleting the corresponding edges can be done in constant time. We can perform at most k recursive calls to `findTriangles` as the value of k decreases by one in each round. Hence, the running time of the process in order $\mathcal{O}((32k^3)^k) = \mathcal{O}(2^{3klog(\sqrt[3]{32}k)}) \leq \mathcal{O}(2^{3k\log(3.2k)})$.

Step 5 and Step 6: Steps 5 and 6 be carried out in constant time.

Therefore, the total running time of the algorithm is $\mathcal{O}(|V|^4 + |E| + 2^{3k}(|V|^2 + |E|) + 2^{3k\log(3.2k)}) = \mathcal{O}^*(2^{3k\log(3.2k)})$. $\qquad\qquad\qquad\square$

It is to be noted that single exponential running time can be achieved for both EDGE-DISJOINT TRIANGLE PACKING and one of the problems in Chapter 5, s-STAR PACKING. The algorithm to obtain this running time uses a color coding technique, also known as hashing [AYZ95]. The idea is to use dynamic programming on a coloring of the edges of the graph. For more details on this type of technique for packing problems we refer the reader to [FK+04].

7

MAX INTERNAL SPANNING TREE

MOTIVATION

As we did in Chapter 3 with MAX LEAF SPANNING TREE, this chapter addresses a problem concerning spanning trees. In this case, instead of maximizing the number of leaves, we try to maximize the number of internal vertices. This problem is called MAX INTERNAL SPANNING TREE. A natural parameterization of this problem is formally defined as follows:

> MAX INTERNAL SPANNING TREE
>
> Instance: A connected graph $G = (V, E)$ and a positive integer k
>
> Parameter: k
>
> Question: Does G have a spanning tree with at least k internal vertices?

The problem is NP-complete as HAMILTONIAN PATH can be considered a special case of MAX INTERNAL SPANNING TREE by making $k = |V| - 2$ and HAMILTONIAN PATH is NP-complete [Karp72].

The reduction rules and the proofs to structural claims in this chapter are joint work with C. Sloper and have been reported in [PS03]. The proof of Reduction Rule 2 corrects the one appearing in [PS03].

REDUCTION RULES

A vertex $u \in V$ is a *common neighbor* of a pair of distinct vertices $\{w_1, w_2\}$ if both (u, w_1) and (u, w_2) are in $E(G)$.

Let T be a spanning tree with k internal vertices. We say that a leaf in T is *essential* to T if its removal makes T have $k - 1$ internal vertices. Any other leaf in T will be termed *non-essential*. Note that there are at most k essential leaves in T.

Reduction Rule 1 *Let $G = (V, E)$ be a connected graph such that there exist two vertices u and v of degree one. Let w be the common neighbor of $\{u, v\}$ in G. Let $G' = G \smallsetminus v$. Transform (G, k) into (G', k).*
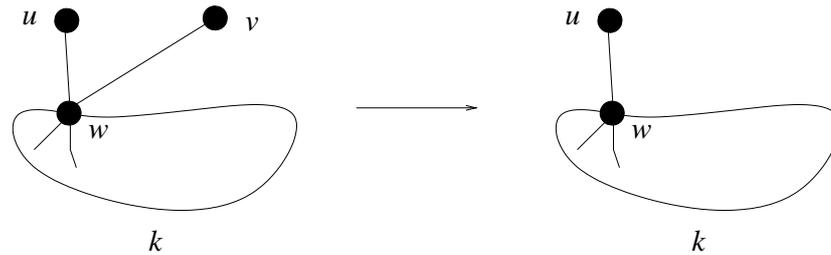
Figure 7.1: Reduction Rule 1 for MAX INTERNAL SPANNING TREE.

Soundness Lemma 1 Reduction Rule 1 is sound.

Proof of Soundness Lemma 1.

($\Leftarrow$:) Assume G' has a spanning tree with k internal vertices. Then G has a spanning tree with k internal vertices since adding a vertex of degree one cannot decrease the number of internal vertices.

($\Rightarrow$:) Assume G has a spanning tree with k internal vertices. Then w is an internal vertex since u and v must always be leaves. In G' the vertex w is still an internal vertex because u can still be a leaf. Thus the number of internal vertices in a spanning tree in G' is not affected by the deletion of v. $\square$

Reduction Rule 2 *Let $G = (V, E)$ be a connected graph. Let J be an independent set in G. Let $u \in J$ and $|N(u)| \geq 2$. Assume that for every pair of vertices $p = \{v, v'\}$ in $N(u)$ there exists a set S_p of $2k + 1$ vertices in J which are common neighbors to v and v'. Assume that for every two different pairs p and p', $S_p \cap S_{p'} = \varnothing$ (see Figure 7.2). Let $G' = G \smallsetminus u$. Transform (G, k) into (G', k).*
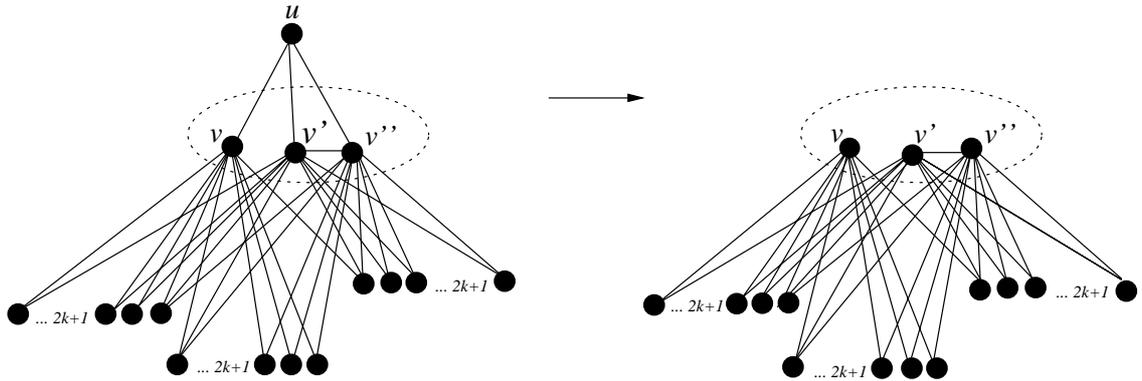
Figure 7.2: Vertex u is common neighbor to the pairs $\{v,v'\}$, $\{v',v''\}$ and $\{v,v''\}$

Soundness Lemma 2   Reduction Rule 2 is sound.

Proof of Soundness Lemma 2.

($\Leftarrow$:) Assume (G', k) is a YES-instance for MAX INTERNAL SPANNING TREE. If G' has a k-internal spanning tree T' then G has a k-internal spanning tree T by extending T' to form a spanning tree for G. The introduction of u does not decrease the number of internal vertices and thus (G, k) is a YES-instance for MAX INTERNAL SPANNING TREE.

($\Rightarrow$:) Assume (G, k) is a YES-instance for MAX INTERNAL SPANNING TREE. Let T be a spanning tree in G with k internal vertices. Two cases arise:

1. Vertex u is a leaf in T.

1.1. The parent in T of u, vertex z, has degree greater than two in T. Then u is a non-essential leaf in T and $T \smallsetminus u$ is a spanning tree with k internal vertices for G'. Thus (G', k) is a YES-instance for MAX INTERNAL SPANNING TREE.

1.2. The parent in T of u, vertex z, has degree two in T. By our assumption we know that z has at least $2k + 1$ other neighbors in J. If at least k of these neighbors are internal vertices in T, then u is a non-essential leaf in T, as its removal will make z a leaf, but there are still k internal vertices in T. Otherwise, there are at least $k + 1$ leaves among the $2k + 1$ neighbors of z. Since at most k of them can be essential, there is at least one vertex w which is a non-essential leaf in T. Let the parent of w in T be z'. Vertex $z' \neq z$ as otherwise the degree of z in T would not be two. Modifying T by deleting (w, z') from T and adding (w, z), makes z be neighbor of more than two vertices in T. Case 1.1 now applies.

2. Vertex u is internal in T.

   Since vertex u is internal, $deg(u) = i \geq 2$ in T. We will show that it is always possible to modify T to obtain a spanning tree T^* with k internal vertices, where u has degree $i - 1$ in T^*. We can repeatedly apply this procedure until u has degree one in T^* and Case 1 applies.

   Consider any two vertices $x, y \in N(u)$ in T. Vertices x and y have at least $2k + 1$ common neighbors in J. Either there are more than k internal vertices among these $2k + 1$ common neighbors, or there are at most k internal vertices.

   2.1. If there are at most k internal vertices, then there are at least $k + 1$ leaves among the $2k + 1$ common neighbors of x and y. Since at most k of them can be essential, there is at least one vertex w which is a non-essential leaf in T. Let the parent of w in T be z. Three

subcases arise:

2.1.1. Vertex $z \notin \{x, y\}$. Then, by removing (x, u) and (w, z) from the spanning tree and adding $(x, w), (w, y)$, we obtain a spanning tree T^* with at least k internal vertices where u is of degree $(i - 1)$ in T^*.

2.1.2. Vertex $z = x$. Then, by removing (x, u) and adding (y, w) from T we obtain a spanning tree T^* with at least k internal vertices where u is of degree $(i - 1)$ in T^*.

2.1.3. Vertex $z = y$. Then, by removing (x, u) and adding (x, w) we obtain a spanning tree T^* with at least k internal vertices where u is of degree $(i - 1)$ in T^*.

2.2. If there are more than k internal vertices, let w be any of them. Let x' and y' be two neighbors in T of w. Since J is an independent set in G, vertices x' and y' are not in J. Also, $\{x, y\} \neq \{x', y'\}$ as otherwise T would contain a cycle. Two subcases arise:

2.2.1. One vertex out of $\{x', y'\}$ is in $\{x, y\}$. Then, without loss of generality assume $y = y'$. Then, by removing (x, u) from T and adding (x, w) we obtain a spanning tree T^* where w is still internal and u is of degree $(i - 1)$ in T^*.

2.2.2. No vertex out of $\{x', y'\}$ is in $\{x, y\}$. Then, there must be a path P in T joining u and w. If P goes through y, then by removing (x, u) from T and adding (x, w) we obtain a spanning tree T^* where w is still internal and u is of degree $(i - 1)$ in T^*. If P does not go through y, then by removing (y, u) from T and adding (y, w) we obtain a spanning tree T^* where w is still internal and u is of degree $(i - 1)$ in T^*.

With Cases 1 and 2, we have proved that T can be modified to obtain a spanning tree T^* where u is a non-essential leaf. Non-essential leaves can be

removed without making (G, k) become a NO-instance. $\square$

Reduction Rule 3 *Let u and v be two adjacent vertices of degree greater than one in G and $(u, v) \in E(G)$ be a bridge in G. Let $G' = G_{[u \leftrightarrow v]}$, the graph obtained by merging u and v into a new vertex $u \leftrightarrow v$. Transform (G, k) into $(G', k - 1)$.*
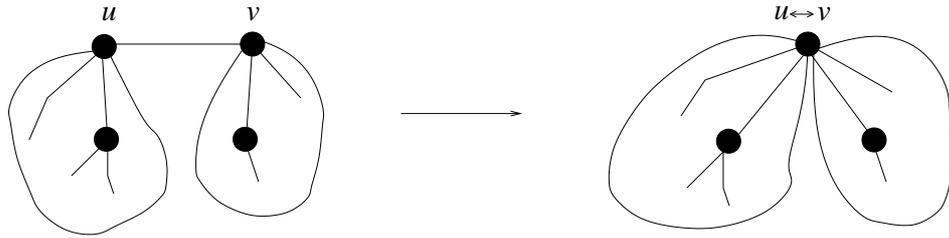
Figure 7.3: Reduction Rule 3 for MAX INTERNAL SPANNING TREE.

Soundness Lemma 3 Reduction Rule 3 is sound.

Proof of Soundness Lemma 3.

$(\Rightarrow:)$ If (u, v) is a bridge then both u and v must be internal to any spanning tree T in G. Thus merging u and v into a single vertex will decrease the number of internal vertices by one, making $(G', k - 1)$ a YES-instance for MAX INTERNAL SPANNING TREE.

$(\Leftarrow:)$ Let T' be a spanning tree with $k - 1$ internal vertices in $G' = G_{[u \leftrightarrow v]}$. The vertex $u \leftrightarrow v$ is an articulation vertex in G' and therefore is always an internal vertex of T'. If we expand this articulation vertex to create G, the edge joining the resulting two vertices will be a bridge and thus both its endpoints will be internal to any spanning tree T in G. The spanning tree T has one more internal vertex than T' and thus (G, k) is a YES-instance for MAX INTERNAL SPANNING TREE.

$\square$

KERNELIZATION AND BOUNDARY LEMMAS

In our analysis we only use Reduction Rules 1 and 2 and thus an instance is considered *reduced* when these rules are no longer applicable. We have included Reduction Rule 3 since it may also be of interest from a practical point of view.

Given a spanning tree T of a graph G we say that two leaves $x, y \in T$ are *in conflict* if $(x, y) \in E(G)$.

The algorithm in Table 7.1 describes a polynomial time preprocessing algorithm to be applied to any instance (G, k) of MAX INTERNAL SPANNING TREE.

Lemma 7.1 *(Algorithmic Boundary Lemma) Let G be a connected graph. If $|V(G)| > k^3$ then the preprocessing algorithm in Table 7.1 will either decide the instance (G, k) or it will reduce it.*

Figure 7.4 illustrates the partition of the vertex set obtained by the preprocessing algorithm described in Table 7.1. Note that the thicker lines indicate the edges in the spanning tree T.
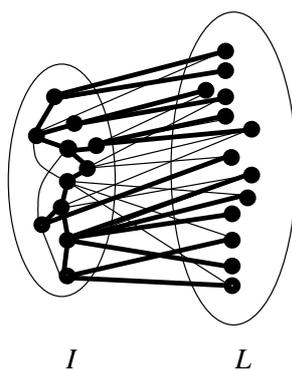
Figure 7.4: Example of structure after preprocessing algorithm has been run.

Step 1. Compute a spanning tree T. Denote by I
         the set of internal vertices in T and by L
         the set of leaves. While there exist two
         leaves x and y in L that are in conflict do:
  1.a If x and y are in conflict and $z \in I$,
       the parent of x, has degree greater than
       two in T, construct a new spanning tree
       T' using the edge (x,y) instead of (x,z).
       Since z is now of degree at least two in
       T' we decrease the number of leaves by
       one.
  1.b If x and y are in conflict, both their
       parents have degree two in T, and there
       exists a vertex x' on a path in T from x
       to y that has degree greater than two in
       T, then let x'' be the predecessor of x' on
       that path. Construct a new spanning tree
       T' using the edge (x,y) instead of (x'',x').
       Since x' is now of degree at least 2 in
       T' we decrease the number of leaves by at
       least one.
  1.c If x and y are in conflict and all ver-
       tices on a path in T from x to y have
       degree two in T, since T is spanning,
       then T is a Hamiltonian path in G.
    1.c.1 If $|V| \geq k + 2$ then answer YES and
           halt.
    1.c.2 Otherwise answer NO and halt.
         (We will argue in Lemma 7.1 that after this
         step has been run, if the instance has not yet
         been decided, the set L is an independent set
         in G).
Step 2. If there exist two vertices $u,v \in L$ of degree
         one in G such that $N(u)=N(v)$, then reduce using
         Reduction Rule 1.
Step 3. Let L_+ be the set of vertices in L of degree
         greater than 1 in G. For every $w \in L_+$ do: If
         for every pair $\{u,v\}$ of vertices in $N(w)$ there
         exist $2k+2$ distinct common neighbors to $\{u,v\}$ in
         L_+ then reduce using Reduction Rule 2.

Table 7.1: Preprocessing algorithm for MAX INTERNAL SPANNING TREE.

Proof of Lemma 7.1. Assume in contradiction that $|V(G)| > k^3$, but that the algorithm has neither decided the instance (G, k) nor reduced it.

After the preprocessing algorithm has been run, we have a spanning tree T. Consider the following partition of $V(G)$:

- The subset I of internal vertices of T,

- The subset L of leaves of T.

Structural Claims

Claim 1 *The set I has at most $k - 1$ vertices.*

> *Proof of Claim 1.* The claim holds as otherwise we would have a YES-instance for MAX INTERNAL SPANNING TREE contradicting the claim that the instance has not been decided. $\square$

Claim 2 *If L has more than $k - 1$ vertices of degree one then we can reduce using Reduction Rule 1.*

> *Proof of Claim 2.* Note that all vertices in I have degree at least two in G. If there exist more than $k - 1$ vertices of degree one, they would be in L and then at least one vertex in I would have two neighbors of degree one in G. Thus there exist two vertices $u, v \in L$ such that $N(u) = N(v)$ and we can reduce using Reduction Rule 1. $\square$

Claim 3 *The set L is an independent set in G.*

Proof of Claim 3. Assume in contradiction that there exist conflicts between the leaves in L. If the spanning tree T generated in Step 1 of the preprocessing algorithm has conflicts between its leaves, Steps 1.a and 1.b will modify T to obtain a spanning tree with fewer leaves. Thus, Step 1 terminates only when L is an independent set or when T is a Hamiltonian path is G. In this case Step 1.c would answer YES or NO, contradicting the assumption that the instance has not been decided. $\square$

We say that $w \in L$ is an *L-link* for two vertices $u, v \in I$ if w is a common neighbor to u and v. Let P_I be the set of all pairs of vertices in I. Each pair of vertices in P_I determine a *L-link position* in L. An *L*-link position determined by a pair of vertices u and v in I *contains* a vertex w if $w \in L$ is an *L-link* for u and v.

Claim 4 *There are at most $\frac{(k-1)(k-2)}{2}$ possible L-link positions in L.*

Proof of Claim 4. Since there are at most $k - 1$ vertices in I by Claim 1, there are at most $\binom{k-1}{2}$ pairs of vertices in P_I, each one determining one L-link position. $\square$

Claim 5 *If each L-link position in L contains at least $2k + 2$ L-link vertices, then we can use Reduction Rule 2.*

Proof of Claim 5. Let u be a vertex in L. Since every L-link position contains at least $2k + 2$ vertices, in particular for every pair $\{v, v'\}$ of vertices in $N(u)$ there exist u and $2k + 1$ other distinct vertices in L which are common neighbors to the pair $\{v, v'\}$. Since L is an independent set, vertex u is thus under the hypotheses of Reduction Rule 2, and (G, k) can be reduced. $\square$

Claim 6 *If L_+ has more than $k^3 - 2k^2 - k + 2$ vertices then we can reduce using Reduction Rule 2.*

Proof of Claim 6. Assume in contradiction that there existed more than $k^3 - 2k^2 - k + 2$ vertices in G.

By Claim 5, there is at least one L-link position that contains less than $2k + 2$ vertices. Let $\{u, u'\} \in P_I$ be such a pair.

Mark[1] the pair $\{u, u'\} \in P_I$ and all the vertices in the L-link position determined by $\{u, u'\}$.

Consider now the set $P_I' = P_I \smallsetminus \{u, u'\}$. If all pairs in P_I' determine L-link positions containing more than $2k + 2$ unmarked vertices, the argument in Claim 5 applies to P_I' and the L-links that are left unmarked. Hence, (G, k) can be reduced by Reduction Rule 2, contradicting the claim that no reduction has taken place. Thus, there is at least one unmarked pair determining an L-link position that contains less than $2k + 2$ unmarked vertices. Assume that the pair is $\{v, v'\}$. Mark the pair $\{v, v'\}$ and all the L-links in the L-link position determined by v and v'. Make $P_I'' = P_I' \smallsetminus \{v, v'\}$. Recursively apply this process until all pairs are marked. Since, by Claim 4, there are at most $\frac{(k-1)(k-2)}{2}$ such pairs, no more than $(2k + 2)\frac{(k-1)(k-2)}{2} = k^3 - 2k^2 - k + 2$ vertices of degree greater than one can exist in the set L. $\qquad\square$

The total size $|V(G)| = |I| + |L|$. By Claim 2 we know that there are at most $k - 1$ vertices of degree one in L, as otherwise we could reduce using Reduction Rule 1, contradicting the claim that no reduction has taken place. By Claim 6 there are at most $k^3 - 2k^2 - k + 2$ vertices of degree greater than one in L as otherwise we could reduce using Reduction Rule 2, contradicting the claim that no reduction has taken place. As there are at most $k - 1$ vertices in I, $|V(G)| \leq k^3 - 2k^2 - k + 2$. This contradicts the assumption that $|V(G)| > k^3$ since $k^3 - 2k^2 - k + 2 < k^3$ for all positive values of k, concluding the proof of Lemma 7.1. $\qquad\square$

---

[1] By *marking* the pair $\{u, u'\}$ what is meant is marking the 2-element set $\{u, u'\}$.

Lemma 7.2 *(Algorithmic Kernelization Lemma) Any instance (G, k) can be reduced to a problem kernel of size at most k^3.*

Proof of Lemma 7.2. This follows from Lemma 7.1, as we can run the preprocessing algorithm until it fails to reduce G.   □

ALGORITHM AND RUNNING TIME ANALYSIS

In the previous section we obtained a kernel for the problem. In Table 7.2 we provide an algorithm to solve MAX INTERNAL SPANNING TREE.

```
Step 1.  Apply the preprocessing algorithm in Table 7.1
         to an instance (G, k).
Step 2.  If |V(G)| > k³ then answer YES and halt.
Step 3.  Else find all possible sets S of size k in
         (G, k).
Step 4.  For each one of the sets in Step 3 check if the
         vertices in V(S) can be used as internal ver-
         tices for a spanning tree of G in a brute force
         manner.
Step 5.  If the answer is YES for one of the subsets in
         Step 3, answer YES and halt.
Step 6.  Else answer NO and halt.
```

Table 7.2: FPT algorithm to solve MAX INTERNAL SPANNING TREE.

Lemma 7.3 MAX INTERNAL SPANNING TREE *can be solved in time $\mathcal{O}^*(2^{\frac{7}{2}k \log k})$.*

Proof of Lemma 7.3.

Step 1: We must analyze the running time of the preprocessing algorithm in Table 7.1:

We can use breadth-first search to obtain a spanning tree T. This can be done in time $\mathcal{O}(|V| + |E|)$ [CLRS01]. In T, let I be the set of internal vertices and L the set of leaves. If the leaves in T are not independent we must repair the conflicts. Each conflict can be

detected in time $\mathcal{O}(|E|)$ and repaired in time $\mathcal{O}(|V|)$. Note that at the end of Step 1 of the preprocessing algorithm, the set of leaves L is independent.

Reduction Rule 1 (Step 2 of the preprocessing algorithm) can be performed in time $\mathcal{O}(|V|^2)$ as finding vertices of degree one can be done in time $\mathcal{O}(|V|^2)$ and deleting in constant time.

To reduce the number of vertices in L using Reduction Rule 2, we proceed as explained in Step 3 of the preprocessing algorithm. For each vertex $u \in L$ we count the number of vertices that are L-links for each pair of neighbors of u. If all pairs have more than $2k+1$ L-links we can remove u. As there are less than $\frac{(k-1)(k-2)}{2}$ such pairs, this can be done in time $\mathcal{O}(k^2 \cdot |V|)$.

Thus, reducing the instance to a cubic kernel can be done in polynomial time on the size of the instance.

Step 2: This step can be done in linear time in the size of $V(G)$.

Step 3: To check if there is a spanning tree with k internal vertices we test every possible subset S of size k in the kernel to construct a tree T'. The number of k-sets in G is bounded by the following function:

$$\binom{k^3}{k} = \frac{k^3(k^3 - 1)\cdots(k^3 - k + 1)}{k!} \leq \frac{(k^3)^k}{k^{k/2}} = k^{\frac{5}{2}k} = 2^{2.5k\log k}$$

Step 4: If T' has k vertices and for all leaves in T' we can find a unique vertex in $V(G) \smallsetminus V(T')$, then T' can be extended to a spanning tree T which has at least k internal vertices. To do this we try every possible construction of a spanning tree T' in S. By Cayley's formula, there are at most k^{k-2} different spanning trees in S. Each tree can be found in time $\mathcal{O}(|E(S)| + |V(S)|) = \mathcal{O}(k^2)$ [CLRS01]. Thus all trees in S can be found in time bounded by $k^2 \cdot k^{k-2} = k^k = 2^{k\log k}$. Then for each leaf in T' at least one vertex in the remaining kernel can be designated to ensure that vertex is internal to T. This

is equivalent to testing if there is a bipartite matching between the leaves of T' and the remaining kernel $V(G) \setminus V(S)$. Checking if there is a bipartite matching can be done in time $\mathcal{O}(\sqrt{|V|}|E|)$ [HK73]. There are no more than k^4 edges between the leaves in T' and $V(G) \setminus V(S)$ giving us a total running time of $\mathcal{O}(k^{\frac{11}{2}})$ to check if there is a matching. Thus for each k-set S we can verify in time bounded by $2^{k \log k} \cdot k^{\frac{11}{2}}$ if S determines the k internal vertices of a solution spanning tree for G.

The total running time of Steps 3 and 4 is bounded by $\mathcal{O}(2^{\frac{7}{2}k \log k} \cdot k^{\frac{11}{2}})$.

Step 5 and Step 6: Steps 5 and 6 can be carried out in constant time.

The total running time of the algorithm is thus $\mathcal{O}(|V||E| + |V|^2 + k^2|V| + 2^{\frac{7}{2}k \log k} \cdot k^{\frac{11}{2}}) = \mathcal{O}^*(2^{\frac{7}{2}k \log k})$ as claimed in the lemma.   □

MINIMUM MAXIMAL MATCHING

MOTIVATION

A maximal matching is a matching such that every edge which is not part of the matching shares a common endpoint with some edge in the matching. Finding a minimum maximal matching, the MINIMUM MAXIMAL MATCHING problem, is NP-complete [GY80]. A natural parameterization of the problem, taking as a parameter the number edges allowed in the matching, is formally defined as follows:

MINIMUM MAXIMAL MATCHING

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

Parameter: k

Question: Is there a subset $E' \subseteq E$ of size at most k, such that no two edges
             in E' share a common endpoint and every edge in $E \smallsetminus E'$ shares a
             common endpoint with some edge in E'?

E' is called a *matching* in $E(G)$ and the edges in E' are called *matched edges*. It is possible that some vertices will not be endpoints of any edge in E'. We will call those vertices which are endpoints to an edge in E' *vertices in the matching* or *matched vertices*.

Yannakakis and Gavril [GY80] prove that the related EDGE DOMINATING SET problem, which asks for a set $E' \subseteq E$ of k or fewer edges such that every edge in E' shares at least one endpoint with some edge in E', is also NP-complete, even for planar bipartite graphs with maximum degree three.

MINIMUM MAXIMAL MATCHING has been studied from many different angles, in many cases when considering the MINIMUM MAXIMAL FLOW problem [Ski97]. It has a 2-approximation algorithm (any maximal matching approximates within a factor of 2) and it is APX-complete [AC+99]. Heuristics [FNU01] and randomized algorithms [Zit99] have also been proposed for MINIMUM MAXIMAL MATCHING. As a lower bound, M. Chlebík, J. Chlebíková [CC03] show that it is NP-hard to approximate the solution of the problem to within any constant factor smaller than $\frac{7}{6}$.

Restrictions to MINIMUM MAXIMAL MATCHING have been studied in the field of parameterized complexity. There are FPT algorithms for this problem for graphs excluding single-crossing graphs as minors, graphs of bounded genus, $K_{3,3}$ and K_5 minor-free graphs [DHT04] and in general for H-minor-free graphs [DFHT03].

REDUCTION RULES

In the case of MINIMUM MAXIMAL MATCHING we introduce a generalization of the *double crown* decomposition defined in Chapter 5. Here each vertex in H has k vertices from C assigned to it (as opposed to only two as in the case of double crowns). An illustration of this type of crown is shown in Figure 8.1 for the case $k = 3$. The dashed lines indicate how each vertex in H is assigned three vertices in C.
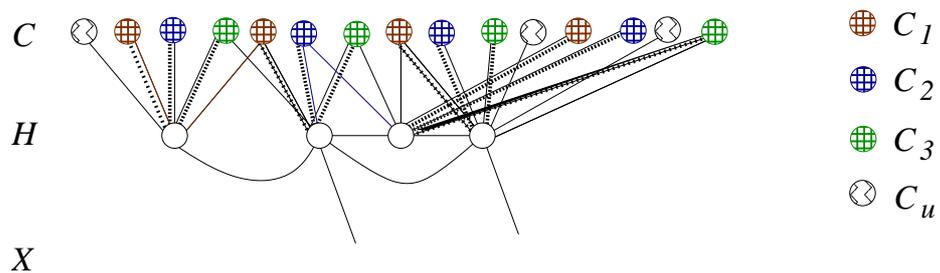

Figure 8.1: Example of k-spike crown with $k = 3$.

This modeled crown decomposition is called k-*spike crown decomposition* and is formally defined as follows:

Definition 8.1 *A k-spike crown decomposition of a graph $G = (V, E)$ is a partition of the vertices of the graph into three non-empty sets C, H and X with the following properties:*

1. *H is a separator in G such that there are no edges in G between vertices belonging to C and vertices belonging to X.*

2. *C is an independent set in G.*

3. *$C = C_u \cup C_1 \cup C_2 \cup \ldots \cup C_k$ is an $(k+1)$-partition of C with $C_u \neq \varnothing$.*

4. *For all $1 \leq i \leq k$ $|C_i| = |H|$, and there is a perfect matching M_i between C_i and H.*

5. *A one-to-many mapping f assigns to every $v \in H$ exactly k distinct vertices $\{v_1, v_2, \ldots, v_k\} \in C$ such that $(v, v_i) \in M_i$ for $1 \leq i \leq k$.*

Before presenting the k-*spike crown* reduction rule, we state the following observation:

Proposition 8.1 *Let $I = \{u_1, u_2, \ldots, u_l\}$ an independent set in G of size $l > k$. If (G, k) is a YES-instance for MINIMUM MAXIMAL MATCHING, and there exists a vertex $u \in V$ such that $N(u) \supseteq I$, then u is a vertex in any minimum maximal matching of size k.*

Proof of Proposition 8.1. This proposition can be proved by contradiction. If u is not a matched vertex, all its neighbors in I are matched vertices, as otherwise the matching is not maximal. Since $|I| > k$, any matching containing every vertex in I, has size greater than k. Thus, if u is not part of any maximal matching, there is no maximal matching in G of size at most k, and (G, k) is a NO-instance for MINIMUM MAXIMAL MATCHING, a contradiction. $\qquad\square$

Reduction Rule 1 *Let (G, k) be a graph instance that admits a $(k+1)$-spike crown decomposition (C, H, X). Let G' be G after deleting C_u. Transform (G, k) into (G', k).*

Soundness Lemma 1 Reduction Rule 1 is sound.

Proof of Soundness Lemma 1.

$(\Rightarrow:)$ Assume (G, k) is a YES-instance of MINIMUM MAXIMAL MATCHING which admits a $(k+1)$-spike crown decomposition. Assume M is a maximal matching of size k in G. If the vertices in C_u are not endpoints in M, then deleting C_u will not affect the size of any maximal matching. If, to the contrary, there exists a vertex u in C_u which is part of the matching, as C is an independent set in G, u must be matched to a vertex v in H. Hence, a vertex w can be found in the $(k+1)$-spike crown in one of the C_i sets with $i \neq u$. Thus we can construct a new matching which does not contain the edge (u, v) but contains (v, w) instead. The matching M has size k proving that the deletion of C_u can never transform any YES-instance (G, k) of MINIMUM MAXIMAL MATCHING into a NO-instance.

$(\Leftarrow:)$ Conversely, by Proposition 8.1, every vertex in H is a vertex in the matching as they can always be matched to vertices in C_i with $i = 1, 2, \ldots, k$. Thus, adding any vertex which is only adjacent to vertices in H is not going to transform any YES-instance (G', k) into a NO-instance. $\square$

The following corollary to Lemma 1.3 is used to determine when we can find a $(k+1)$-spike crown decomposition in a graph instance.

Corollary 8.1 *Any graph G with an independent set I, where $|I| > (k + 1)|N(I)|$, has a $(k+1)$-spike crown decomposition (C, H, X), where $H \subseteq N(I)$. Given I, this decomposition can be found in time $\mathcal{O}(|G|^2)$.*

Proof of Corollary 8.1. Let G be a graph with an independent set $I \subseteq V(G)$ such that $(k+1)|N(I)| < |I|$. An auxiliary graph G' is constructed from G, by adding for every vertex $v \in N(I)$ vertices $\{v_1, v_2, \dots, v_k\} \in C'$ with $N(v_i) = N(v)$ for all i. By Lemma 1.3, G' has a crown decomposition (C', H', X'), with $C' = C$, $H' \subseteq N_{G'}(I)$, since $|H'| < |N_{C'}(H')|$ and C' is an independent set. We use this crown to construct a $(k+1)$-spike crown (C, H, X) in G.

First observe that $v \in H'$ if and only if $v_i \in H'$ for all i. Assume in contradiction that there exists a crown decomposition such that $v \in H'$ but $v_i \notin H'$. The vertex v must be matched to some vertex u in C'. Let v' be one of the copies of v in the auxiliary graph. Since $N(v) = N(v')$ we have that v' cannot be in C' as it would contradict that C' is an independent set in G. Also, for all i, v_i is not in X' as that would contradict that H' is a separator. Thus v_i must be in H', contradicting the assumption.

Based on this observation, the result follows as H' consists of groups of vertices, a vertex and its k copies. Each set $S = \{v, v_1, \dots, v_k\} \in H'$ has size $k+1$ and forms a perfect matching to $k+1$ vertices $\{u, u_1, \dots, u_k\}$. In G, let v be in H and let f be the function assigning v to the vertices $\{u, u_1, \dots, u_k\}$ in the matching in G'. Do this for every $v \in H'$. This forms a $(k+1)$-spike crown in G.

Constructing the graph G' can be done in quadratic time on the size of G. By Lemma 1.3 we know that a crown decomposition can be found in a graph time $\mathcal{O}(|E| + |V|)$, given I. Thus a $(k+1)$-*spike crown decomposition* can be found in quadratic time on the size of G, given I. $\qquad\qquad\square$

KERNELIZATION AND BOUNDARY LEMMAS

Before giving the preprocessing algorithm, it is important to note that if there exists a maximal matching M in G of size $|M| > 2k$ then (G, k) is a NO-instance for MINIMUM MAXIMAL MATCHING. Assume we have more than $2k$ independent edges in G. Any edge in a minimum maximal matching could cover at most two of those edges and thus k

edges would always leave at least one edge outside making it impossible for (G, k) to be a YES-instance of MINIMUM MAXIMAL MATCHING.

In Table 8.1 we describe a polynomial time preprocessing algorithm which we apply to an instance (G, k) of MINIMUM MAXIMAL MATCHING.

```
Step 1.  Compute an arbitrary maximal matching M in G.
         Let O = V ∖ V(M).
         (We will argue in Lemma 8.1 that the maximality
         of M implies that O is an independent set in
         G).
Step 2.  Consider the matching M.
          2.a  If |M| ≤ k answer YES and halt.
          2.b  If |M| > 2k answer NO and halt.
Step 3.  If |O| > (k+1)|N(O)| by Corollary 8.1 there is
         a (k+1)-spike crown.  Reduce using Reduction
         Rule 1.
```

Table 8.1: Preprocessing algorithm for MINIMUM MAXIMAL MATCHING.

Lemma 8.1 *(Algorithmic Boundary Lemma) If $|V(G)| > 4k(k + 2)$ then the preprocessing algorithm will either decide the instance (G, k) or it will reduce it.*

Proof of Lemma 8.1. Assume in contradiction to Lemma 8.1 that $|V(G)| > 4k(k + 2)$, but that the algorithm has neither decided the instance (G, k) nor reduced it.

After the preprocessing algorithm has been run, $V(G)$ is partitioned as follows:

- $V(M)$, the vertices in the maximal matching M for G.

- $O = V \smallsetminus V(M)$, the vertices not present in M.

Structural Claims

Claim 1 *The set O is an independent set in G.*

Proof of Claim 1. If there exists an edge in $\langle O \rangle$, then M is not maximal. □

Claim 2 *The size of O, $|O| \leq 4k(k+1)$.*

Proof of Claim 2. Assume in contradiction that $|O| > 4k(k+1)$. Since the size of M is at most $2k$, then the number of vertices in M is at most $4k$. As O is an independent set in G, if $|O| > 4k(k+1) \geq |N(O)|(k+1)$ then, by Corollary 8.1, G has a $(k+1)$-spike crown and should have been reduced in Step 3 of the algorithm, contradicting the claim that no reduction has taken place. □

Thus the total size $|V(G)| = |V(M)| + |O| \leq 4k + 4k(k+1) = 4k(k+2)$. This contradicts the assumption that $|V(G)| > 4k(k+2)$. □

Lemma 8.2 *(Algorithmic Kernelization Lemma) Any instance (G, k) can be reduced to a problem kernel of size at most $4k(k+2)$.*

Proof of Lemma 8.2. This follows from Lemma 8.1, as we can run the preprocessing algorithm until it fails to reduce (G, k). By Lemma 8.1 the size is then at most $4k(k+2)$. □

ALGORITHM AND RUNNING TIME ANALYSIS

In the previous section we obtained a kernel for the problem of size $4k(k+2)$ on the number of vertices. In Table 8.2 we provide an algorithm to solve MINIMUM MAXIMAL MATCHING.

Lemma 8.3 MINIMUM MAXIMAL MATCHING *can be solved in time $\mathcal{O}^*(2^{4k(k+2)})$.*

```
Step 1.  Apply the preprocessing algorithm in Table 8.1
         to the instance (G, k).
Step 2.  If |V(G)| > 4k(k + 2) answer NO and halt.
Step 3.  Else find if there exists a maximal matching of
         size k in G in a brute force manner.
Step 4.  If there is a maximal matching of size k answer
         YES and halt.
Step 5.  Else answer NO and halt.
```

Table 8.2: FPT algorithm to solve MINIMUM MAXIMAL MATCHING.

Proof of Lemma 8.3.

Step 1: The running time of the preprocessing algorithm is bounded by the running time of Reduction Rule 1. This rule gets triggered as soon as we construct a maximal matching in G. Once the rule gets triggered, it can be carried out in quadratic time on the size of G. A maximal matching can be found in time $\mathcal{O}(|E|)$.

Step 2: The running time of Step 2 is linear in $|V|$ as we have to check all vertices in the graph.

Step 3: We find if there exists a maximal matching of size k in G in a *brute force* manner. First we look for possible candidate matchings. There are $\sum_{i=0}^{2k} \binom{4k(k+1)}{i}$ ways to find a candidate maximal matching of size at most k by choosing at most $2k$ vertices in the graph in all possible ways. An upper bound on the number of candidate matchings is given by the following formula:

$$\sum_{i=0}^{2k} \binom{4k(k+2)}{i} \leq \sum_{i=1}^{4k(k+2)} \binom{4k(k+2)}{i} = 2^{4k(k+2)}.$$

For each one of these sets of vertices we must check if the subgraph the set induces has a perfect matching. This can be done in time $\mathcal{O}\big(\sqrt{4k} \cdot (4k^2)\big)$ using the algorithm in [MV80].

Then we must check if each matching is maximal. This can be achieved by checking if all remaining vertices are independent. We do this by testing in linear time if the subgraph they induce has no edges.

Step 4: Step 4 can be performed in constant time.

The total time required to perform all steps is $\mathcal{O}(n^2 + \sqrt{4k} \cdot (4k^2) \cdot 2^{4k(k+2)}) = \mathcal{O}^*(2^{4k(k+2)})$.

$\square$

9

CONCLUSIONS

9.1 SUMMARY

With the *method of extremal structure*, we have introduced a systematic approach to developing FPT algorithms. This approach to algorithm design is based on kernelization techniques and extremal combinatorics.

When *kernelizing*, the aim is to transform in polynomial time instances (I, k) of a problem $\mathcal{P}$ into smaller instances (I', k'). These transformed instances have the property that (I, k) is a YES-instance for $\mathcal{P}$ if and only if (I', k') is a YES-instance for $\mathcal{P}$. Once the instances are no longer susceptible to these transformations (the instances are *reduced*), two lemmas, the kernelization and boundary lemmas, are invoked to prove that $|I'| \leq f(k)$ and the problem is kernelizable.

The method operates following a paradigm presented by two lemmas, the kernelization and boundary lemmas. The boundary lemma is used to find theoretical bounds on the maximum size of an instance reduced under an adequate set of reduction rules. The kernelization lemma is invoked to decide the instances which are larger than $f(k)$ for some function f depending only on the parameter k.

The first aim of the *method of extremal structure* is to provide a systematic way to discover reduction rules for fixed-parameter tractable problems. The second is to devise an analytical way to find theoretical bounds for the size of kernels for those problems.

The *method of extremal structure* can be applied to many parameterized decision problems, but sometimes it is useful to modify the approach to the kernelization and boundary lemmas in order to obtain better results.

In the following subsection we explain the general *kernelization and boundary lemmas* and how to modify them to suit particular situations in which the general form of the *method of extremal structure* does not apply in a straightforward manner. For example, sometimes we will need to algorithmically preprocess the instance to identify or apply certain reduction rules. Then, we use the *algorithmic boundary and kernelization lemmas*. The general boundary and kernelization lemmas are tailored to solve maximization problems. Hence, the approach to minimization problems differs from that of maximization problems. When we deal with minimization problems we employ the algorithmic boundary and kernelization lemmas adapted to this type of problems.

9.1.1 GENERAL BOUNDARY AND KERNELIZATION LEMMAS

The *general boundary lemma* is used to find a bound $g(k)$ on the maximum size of a reduced instance (I, k). In the case of maximization problems it takes the following form, for some function g:

> **Lemma.** *(Boundary Lemma) Let (I, k) be reduced and let (I, k) be a* YES-*instance and $(I, k+1)$ be a* NO-*instance for $\mathcal{P}$. Then the size of I is less than* $g(k)$.

To prove such a lemma, we set up a list of reduction rules that lead to a specific notion of irreducibility and a suitable bounding function $g(k)$.

We prove the boundary lemma by minimum counterexample. The *combinatorial extremal argument* operates in a reduced instance I by assuming in contradiction that (I, k) is a YES-instance for $\mathcal{P}$, $(I, k+1)$ is NO-instance for $\mathcal{P}$ and the size of I is at least $g(k)$. Then:

1. Assume a solution (or 'witness' structure). The solution has size k and is optimal in an extremal sense: no better solutions exist.

   The witness structure is strategically chosen to provide a framework to identify reduction rules.

2. Set inductive priorities that the witness structure above must satisfy. Here the fact that I doesn't have a solution of size $k + 1$ will be implicitly understood as the zeroth inductive priority.

   These inductive priorities often condition the type of reduction rules we will need for the problem.

3. Prove structural claims that hold if all inductive priorities are satisfied and all reduction rules are no longer applicable.

4. Use the structural claims to recognize the need for more inductive priorities, find new reduction rules, identify more structural claims, etc.

5. Stop when all claims are satisfied and the total size of the instance is less than the function $g(k)$, reaching a contradiction.

It is very important to point out the fact that discovering useful reduction rules, setting appropriate inductive priorities and establishing relevant structural claims is a creative art. Usually, when working on a problem, all of the mentioned steps outlined above go hand in hand. Methodologically speaking, as it often happens in mathematics, we do not generate the reduction rules and structural claims from a void. This is one of the reasons why the *method of extremal structure* has proven so powerful: when creating the witness structure it becomes very clear what sort of reduction rules we would need to reduce the size of the instance.[1]

---

[1] Note here that not only the reduction rules provide this specific notion of irreducibility we need for the boundary lemma, but also the witness structure we set in the boundary lemma helps us discover particular reduction rules.

The *general kernelization lemma* is invoked to decide reduced instances of size larger than $f(k)$, for some arbitrary function f. In the case of maximization problems it takes the following form:

> **Lemma.** *(Kernelization Lemma) If (I, k) is an reduced instance of a problem $\mathcal{P}$ and $|I| > f(k)$, then (I, k) is a* YES*-instance for the problem.*

Once the boundary lemma is proved, the kernelization lemma follows by contradiction. Note that if the function g found in the boundary lemma is monotone increasing then $f = g$ and the proof will always look as follows:

> **Proof.** Assume in contradiction to the stated kernelization lemma that there exists an instance (I, k) of size $|I| \geq f(k)$ which is a NO-instance for $\mathcal{P}$.
>
> Let $k' < k$ be the largest k' for which (I, k') is a YES-instance. By the boundary lemma we know that $|I| < f(k') < f(k)$. This contradicts the assumption.

For an example of how to apply this general version of the *method of extremal structure* the reader is referred to the case of MAX CUT in Chapter 2, MAX LEAF SPANNING TREE in Chapter 3, NONBLOCKER in Chapter 4 and s-STAR PACKING in Chapter 5.1.

9.1.2 ALGORITHMIC BOUNDARY AND KERNELIZATION LEMMAS

The algorithmic version of the *method of extremal structure* operates by first establishing a preprocessing algorithm as sometimes we need to algorithmically preprocess the instance to apply certain reduction rules. This algorithm finds a maximal set which will later act as a witness structure in the algorithmic boundary lemma. Once the instance is preprocessed the reduction rules can be applied and the following lemma holds:

> **Lemma.** *(Algorithmic Boundary Lemma) If $|I| > f(k)$ for some function f, then the preprocessing algorithm will either find a solution for $\mathcal{P}$ or it will reduce I.*

The proof of the algorithmic boundary lemma is by contradiction. We assume that the preprocessing algorithm failed to reduce an instance of size greater than $f(k)$. In the process of proving the lemma we will also have to prove a set of structural claims as we did with the general boundary lemma.

The algorithmic *kernelization lemma* is almost identical to the general kernelization lemma and is invoked to decide preprocessed instances of size larger than $f(k)$, for some function f.

> **Lemma.** *(Algorithmic Kernelization Lemma) Any instance (I, k) of $\mathcal{P}$ can be reduced to a problem kernel of size $f(k)$.*

The proof of this last lemma follows from the boundary lemma, as we can run the preprocessing algorithm until it fails to reduce (I, k).

For an example of how to apply this algorithmic version of the *method of extremal structure* we refer the reader to the case of 2-STAR PACKING in Chapter 5.2, EDGE-DISJOINT TRIANGLE PACKING in Chapter 6 and MAX INTERNAL SPANNING TREE in Chapter 7.

9.1.3 THE METHOD OF EXTREMAL STRUCTURE ON MINIMIZATION PROBLEMS

Minimization problems are special since it is harder to bound the size of the witness structure in the boundary lemma. So far only the MINIMUM MAXIMAL MATCHING problem we present in Chapter 8 has been solved using the *method of extremal structure*.

In a minimization problem the idea is to prove the following kernelization and boundary lemmas:

Lemma. *(**Minimization Boundary Lemma**) Let (I, k) be reduced and let (I, k) be a* YES-*instance and $(I, k-1)$ be a* NO-*instance for $\mathcal{P}$. Then the size of I is less than $f(k)$ for some function f.*

Lemma. *(**Minimization Kernelization Lemma**) If (I, k) is an reduced instance of a problem $\mathcal{P}$ and $|I| > f(k)$, then (I, k) is a* NO-*instance for the problem.*

In all examples considered in this thesis, we assume that for every YES-instance (I, k) of $\mathcal{P}$ there exist a witness structure W which is a subset of elements (either vertices or edges) of I. In the case of maximization problems, we impose a further restriction on I by assuming that is I is a NO-instance for $k + 1$. This restriction ensures that no elements in $I \setminus W$ can be added to W that could make W a YES-instance for $k + 1$, effectively bounding the size of W. The difficulty that arises when trying to find what the bounding function may be, in the case of minimization problems, is that the boundary lemma only states that (I, k) is a YES-instance for $\mathcal{P}$ and $(I, k-1)$ is a NO-instance for $\mathcal{P}$. Thus, the size of the witness structure W can be as large as the entire instance, since there are no restrictions on $(I, k + 1)$.

The key issue to take into consideration when we use the *method of extremal structure* to approach minimization problems is that we must make sure that the minimization problem has a forbidden substructure of bounded size that can be calculated in polynomial time which we can use as a witness structure. In the case of MINIMUM MAXIMAL MATCHING, this bounded witness structure is provided by the fact that if a graph G has a matching of size strictly greater than $2k$, then G will never have a maximal matching of size k. This way, by finding a maximal matching we can bound the size of the witness structure as we showed in Chapter 8.

9.2 RESULTS PRESENTED

This thesis has presented fixed-parameter algorithms to solve the problems MAX CUT, MAX LEAF SPANNING TREE, NONBLOCKER, s-STAR PACKING, EDGE-DISJOINT TRIANGLE PACKING, MAX INTERNAL SPANNING TREE and MINIMUM MAXIMAL MATCHING.

The algorithms have the following running times:

- MAX CUT is solved in time $\mathcal{O}^*(1.414^k)$.

- MAX LEAF SPANNING TREE is solved in time $\mathcal{O}^*(14.25^k)$.

- NONBLOCKER is solved in time $\mathcal{O}^*(3.07^k)$.

- s-STAR PACKING is solved in time $\mathcal{O}^*(2^{5.301k})$ when $s = 2$ and $\mathcal{O}^*(4^{k \log k})$ in the general case.

- EDGE-DISJOINT TRIANGLE PACKING is solved in time $\mathcal{O}^*(2^{3k \log(3.2k)})$.

- MAX INTERNAL SPANNING TREE is solved in time $\mathcal{O}^*(2^{\frac{7}{2}k \log k})$.

- MINIMUM MAXIMAL MATCHING is solved in time $\mathcal{O}^*(2^{4k(k+2)})$.

The results for MAX CUT, NONBLOCKER, s-STAR PACKING, MAX INTERNAL SPANNING TREE and MINIMUM MAXIMAL MATCHING are the best parameterized results to date known for these problems.

MAX LEAF SPANNING TREE has a better parameterized algorithm but no explicit kernel bound is provided for the problem, whereas the result we present in this thesis proves a kernel of size at most $5.75k$, linear in the number of vertices in the graph.

Using the color-coding technique developed in [AYZ95] a single exponential result is possible for EDGE-DISJOINT TRIANGLE PACKING. However, the result presented in

this thesis provides a kernel for this problem of size at most $4k$, linear in the number of edges in the graph.

9.3 FUTURE RESEARCH

We conclude this thesis by mentioning some directions for future research.

The usefulness of FPT algorithms to cope with the intractability of problems arising from large data sets has been proven in numerous occasions [AL+03]. Some of these problems arise in the field of Bioinformatics, for example, in data analysis of Single Nucleotide Polymorphisms (SNPs). Several problems regarding SNPs have been already considered from a parameterized complexity viewpoint [Wer03]. However, none of them have been studied using the *method of extremal structure*.

Problems, suggested by Pablo Moscato, that could be addressed in the future are variations to the HAPLOTYPING problem, MINIMUM INFORMATIVE SUBSET OF SNPs, and PARTITION AND COVERING OF BIPARTITE GRAPHS IN BICLIQUES.

There are two problems related to haplotyping that have been thoroughly studied within the framework of parameterized complexity. These problems are MINIMUM SNP RE-MOVAL (MSR) and MINIMUM FRAGMENT REMOVAL (MFR). Rizzi et al. [RBIL02] proved that MFR is fixed-parameter tractable. Wernicke [Wer03] showed that MFR is parameter-equivalent to VERTEX BIPARTIZATION (is there a subset of vertices of size k in the graph whose removal renders the graph bipartite?) and that EDGE BIPARTIZA-TION (is there a subset of edges of size k in the graph whose removal renders the graph bipartite?) is parameter-preserving reducible to MSR. It is important to point out at this stage that even though MSR remains open in terms of graph equivalences, the EDGE BI-PARTIZATION problem has been recently proven fixed-parameter tractable using iterative compression techniques [GG+05]. Thus, it would be very interesting to prove the equivalence between MSR and EDGE BIPARTIZATION in order to demonstrate the membership in the class FPT of MSR. It is also important to note that using the *method of extremal*

structure, the running time of MFR could be substantially decreased making it feasible for solving large data sets. A close look at the closely related MAX CUT problem indicates that the method can be of significant interest.

THE MINIMUM INFORMATIVE SUBSET OF SNPS has been characterized in terms of its complexity and is shown to be closely related to the FEATURE SET problem. Unfortunately, FEATURE SET has been shown to be $W[2]$-complete [CM03]. However, several related problems dealing with parameterized duals of these problems have been identified whose parameterized complexity is yet unknown.

There is a problem appearing in the field of epitope recognition [HMR01] which is another good candidate for the *method of extremal structure*: TEST COVER. The input of the TEST COVER problem consists of a set of items $S = \{1, \ldots, m\}$, and a collection of tests $T_1, \ldots, T_n \subset S$. A test Tj covers or differentiates the item pair $\{h, i\}$ if either $h \in T_j$ or $i \in T_j$, i.e., if $|T_j \cap \{h, i\}| = 1$. A sub-collection $T \subset T_1, \ldots, T_n$ of tests is a test cover if all item pairs are covered by at least one test in T. In terms of its equivalence to graph optimization problems, the TEST COVER problem has a natural reformulation as a cut covering problem on a complete graph. Items correspond to vertices and item pairs to edges. The objective is to find a minimum sized sub-collection of cuts whose union is the complete edge set. This problem is generally known as SET COVER and is NP-complete and $W[2]$-hard. However, a variation of this problem when we ask the tests to have size at most two has better prospects. It is equivalent to packing paths of length two in a graph with an asymptotically tight performance ratio of 11/8 [BH+3]. This problem is the 2-STAR PACKING problem presented in this thesis and is a very good candidate to be implemented using evolutionary algorithms.

The *method of extremal structure* can be combined with other standard procedures used in FPT algorithm design such as iterative compression, crown reductions, shrinking search trees by dynamic programming, color-coding, dynamic programming on tree decompositions, etc. Also, in the last step, meta-heuristic techniques can be used to speed up the

solutions in cases when the data set is too large. A very good example showing how to combine these two approaches to coping with intractability was recently proposed by Gilmour and Dras [GD05].

In a strictly theoretical or combinatorial sense, still remain as open problems to be tackled with the *method of extremal structure*:

1. Minimization problems such as EDGE EDIT TO DISJOINT CLIQUES, EDGE DELE-TION TO BIPARTITE and EDGE DOMINATING SET. So far only MINIMUM MAX-IMAL MATCHING has been approached using this technique.

2. Non-graph problems such as MATRIX DOMINATION. So far only SET SPLITTING [DFR03] has been approached using this technique.

Additional results of extremal graph theory may improve some of the results in this thesis:

1. NONBLOCKER: Every graph with minimum degree three has a dominating set of size at most $\frac{3|V|}{8}$ [R96].

2. MAX LEAF SPANNING TREE: Every graph with minimum degree four has a spanning tree with at least $\frac{2|V|}{5} + \frac{8}{5}$ leaves [GW92].

Linear sized kernels might be possible for s-STAR PACKING, MAX INTERNAL SPAN-NING TREE and MINIMUM MAXIMAL MATCHING.

Bibliography

[AB+02]   J. Alber, H. Bodlaender, H. Fernau, T. Kloks, and R. Niedermeier. *Fixed Parameter Algorithms for* Dominating Set *and Related Problems on Planar Graphs.* Algorithmica, vol. 33, pages 461–493, (2002).

[AC+04]   F. Abu-Khzam, R. Collins, M. Fellows, M. Langston, W. Suters, C. Symons. *Kernelization Algorithms for the Vertex Cover Problem: Theory and Experiments.* Proceedings of 6th Workshop on Algorithm Engineering and Experiments (ALENEX '04), SIAM Proceedings Series, pages 62–69, (2004).

[AC+99]   G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, M. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation.* Springer, (1999).

[AF+04]   J. Alber, H. Fan, M. Fellows, H. Fernau, R. Niedermeier, F. Rosamond, and U. Stege. *Refined Search Tree Techniques for for Dominating Set on Planar Graphs.* Accepted in Journal on Computer and System Sciences, (2004).

[AFN04]   J. Alber, H. Fernau, and R. Niedermeier. *Parameterized Complexity: Exponential Speedup for Planar Graph Problems.* Journal of Algorithms, vol. 52 (1), pages 26–56, (2004).

[AFN02]   J. Alber, M. Fellows, and R. Niedermeier. *Efficient Data Reduction for* Dominating Set*: a Linear Problem Kernel for the Planar Case.* Proceedings of Skandinavian Workshop on Algorithm Theory (SWAT '02), Springer Lecture Notes in Computer Science, vol. 2368, pages 150–159, (2002).

[AKK95]   S. Arora, D. Karger, and M. Karpinski. *Polynomial Time Approximation Schemes for Dense Instances of NP-hard Problems.* Proceedings of 27th An-

nual Association for Computing Machinery Symposium on Theory of Computing, ACM, pages 284–293, (1995).

[AL+03]   F. Abu-Khazam, M. Langston, P. Shanbhag, and C. Symons. *High-Performance Tools for Fixed-Parameter Tractable Implementations.* Presented in 29th Workshop on Graph Theoretic Concepts in Computer Science, Workshop on Fixed Parameter Tractability, (2003).

[ALS05]   F. N. Abu-Khzam, M. Langston and W. H. Suters. *Fast, Effective Vertex Cover Kernelization: A Tale of Two Algorithms.* Proceedings of ACS/IEEE International Conference on Computer Systems and Applications, Cairo, Egypt, January, (2005).

[AN02]   J. Alber and R. Niedermeier. *Improved Tree Decomposition Based Algorithms for Domination-Like Problems.* Proceedings of the 5th Latin American Theoretical INformatics (LATIN '02), Springer Lecture Notes in Computer Science, vol. 2286, pages 613–627, (2002).

[AYZ95]   N. Alon, R. Yuster, and U. Zwick. *Color-Coding.* Journal of the Association for Computing Machinery, vol. 42(4), pages 844–856, (1995).

[Bak94]   B. Baker. *Approximation Algorithms for NP-Complete Problems on Planar Graphs.* Journal of the Association for Computing Machinery, vol. 41(1), pages 153–180, (1994).

[Baz95]   C. Bazgan. *Schémas d'approximation et Complexité Paramétrée.* Thesis, INRIA, Orsay, France, (1995).

[BBW03]   P. Bonsma, T. Brueggemann, and G. Woeginger. *A Faster FPT Algorithm for Finding Spanning Trees with Many Leaves.* Proceedings of the 28th International Symposium on Mathematical Foundations of Computer Science, Springer Lecture Notes in Computer Science, vol. 2747, pages 259–268, (2003).

[BFR98]   R. Balasubramanian, M. Fellows, and V. Raman. *An Improved Fixed Parameter Algorithm for Vertex Cover*. Information Processing Letters, vol. 65, pages 163–168, (1998).

[BH+3]   K.M.J. De Bontridder, B.V. Halldorsson, M.M. Halldorsson, C. Huskens, J.K. Lenstra, R. Ravi, and L. Stougie. *Approximation Algorithms for the Test Cover Problem*. Mathematical Programming, vol. 98, pages 477–491, (2003).

[BK+01]   R. Bejar, B. Krishnamachari, C. Gomes, and B. Selman. *Distributed Constraint Satisfaction in a Wireless Sensor Tracking System*. Workshop on Distributed Constraint Reasoning, International Joint Conference on Artificial Intelligence (IJCAI '01), (2001).

[BM+02]   R. Bar-Yehuda, M. Halldórsson, J. Naor, H. Shachnai, and I. Shapira. *Scheduling Split Intervals*. Proceedings of the 13th Annual Association for Computing Machinery SIAM Symposium on Discrete Algorithms, pages 732–741, (2002).

[Bod89]   H. Bodlaender. *On Linear Time Minor Tests and Depth-First Search*. Proceedings of First Workshop on Algorithms and Data Structures, Springer Lecture Notes in Computer Science, vol. 382, pages 577–590, (1989).

[Bol78]   B. Bollobás. *Extremal Graph Theory*. Academic Press, (1978).

[Bol98]   B. Bollobás. *Modern Graph Theory*. Springer-Verlag, Graduate Texts in Mathematics, vol. 184, (1998).

[BP96]   V. Bafna and P. Pevzner. *Genome Rearrangements and Sorting by Reversals*, SIAM Journal on Computing, vol. 25(2), pages 272–289, (1996).

[CC03]   M. Chlebík, and J. Chlebíková. *Approximation Hardness of Minimum Edge Dominating Set and Minimum Maximal Matching*. Proceedings of Symposium on Algorithms and Computation (ISAAC '03), Springer Lecture Notes in Computer Science, vol. 2906, pages 415–424, (2003).

[CC97]   L. Cai and J. Chen. *On Fixed-Parameter Tractability and Approximability of NP Optimization Problems*. Journal on Computer and System Sciences, vol. 54, pages 465–474, (1997).

[CC+97]   L. Cai, J. Chen, R. Downey, and M. Fellows. *Advice Classes of Parameterized Tractability*. Annals of Pure and Applied Logic, vol. 84, pages 119–138, (1997).

[CFJ04]   B. Chor, M. Fellows, and D. Juedes. *Linear Kernels in Linear Time, or How to Save k Colors in $\mathcal{O}(n^2)$ Steps*. Proceedings 30th Workshop on Graph Theoretic Concepts in Computer Science (WG '04), Springer Lecture Notes in Computer Science, vol. 3353, pages 257–269, (2004).

[CG04]   S. Chandran, and F. Grandoni. *Refined Memorisation for Vertex Cover*. Proceedings International Workshop on Parameterized and Exact Computation (IWPEC '04), Springer Lecture Notes in Computer Science vol. 3162, pages 27–38, (2004).

[CG+98]   P. Crescenzi, D. Goldman, C. Papadimitriou, A. Piccolboni and M. Yannakakis. *On The Complexity of Protein Folding*. Journal of Computational Biology, vol. 5(3), pages 423–466, (1998).

[CKJ01]   J. Chen, I. Kanj, and W. Jia. *Vertex Cover: Further Observations and Further Improvements*. Journal of Algorithms, vol. 41, pages 280–301, (2001).

[CLRS01]   T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, (2001).

[CM03]   C. Cotta, and P. Moscato. *The k-Feature Set Problem is $W[2]$-complete*. Journal on Computer and System Sciences, vol. 67(4), pages 686–690, (2003).

[Co71]   S. Cook. *The Complexity of Theorem Proving Procedures*. Proceedings of the Third Annual Association for Computing Machinery Symposium on Theory of Computing, pages 151–158, (1971).

[CR01]   A. Caprara and R. Rizzi. *Packing Triangles in Bounded Degree Graphs.* In-
           formation Processing Letters, vol. 84(4), pages 175–180, (2002).

[CT97]   M. Cesati and L. Trevisan. *On the Efficiency of Polynomial Time Approxima-
           tion Schemes.* Information Processing Letters, vol. 64(47), pages 165–171,
           (1997).

[CWY00]   Y. Caro, D. West, and R. Yuster. *Connected Domination and Spanning Trees
           with Many Leaves.* SIAM Journal on Discrete Mathematics, vol. 13, pages
           202–211, (2000).

[DF95a]   R. Downey and M. Fellows. *Parameterized Computational Feasibility.* Feasi-
           ble Mathematics II, pages 219–244, (1995).

[DF95b]   R. Downey and M. Fellows. *Fixed-Parameter Tractability and Completeness
           II: Completeness for $W[1]$.* Theoretical Computer Science, vol. 141, pages
           109–131, (1995).

[DF99]   R. Downey and M. Fellows. *Parameterized Complexity.* Springer-Verlag,
           (1999).

[DF+03]   F. Dehne, M. Fellows, H. Fernau, E. Prieto, and F. Rosamond. *A Descrip-
           tion of the Method of Coordinatized Kernels Illustrated by* NONBLOCKER.
           Manuscript in preparation.

[DFHT03]   E. Demaine, F. Fomin, M. Hajiaghayi, and D. Thilikos. *Fixed-Parameter Al-
           gorithms for the (k;r)-Center in Planar Graphs and Map Graphs.* Proceedings
           of 30th International Colloquium on Automata, Languages and Programming
           (ICALP '03), Springer Lecture Notes in Computer Science, vol. 2719, pages
           829–844, (2003).

[DFR03]   F. Dehne, M. Fellows, and F. Rosamond. *An FPT Algorithm for Set Splitting.*
           Proceedings of 29th Workshop on Graph Theoretic Concepts in Computer

Science, Springer Lecture Notes in Computer Science, vol. 2880, pages 180–191, (2003).

[DFRS04] F. Dehne, M. Fellows, F. Rosamond, and P. Shaw. *Greedy Localization, Iterative Compression, and Modeled Crown Reductions: New FPT Techniques, an Improved Algorithm for Set Splitting, and a Novel $2k$ Kernelization for Vertex Cover*. Proceedings of International Workshop on Parameterized and Exact Computation (IWPEC '04), Springer Lecture Notes in Computer Science, vol. 3162, pages 271–280, (2004).

[DFS99]   R. Downey, M. Fellows, and U. Stege. *Parameterized Complexity: A Framework For Systematically Confronting Computational Intractability*. Contemporary Trends in Discrete Mathematics. AMS-DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 49, pages 49–99, (1999).

[DHT04]   E. Demaine, M. Hajiaghayi, and D. Thilikos. *Exponential Speedup of Fixed-Parameter Algorithms On $K_{3,3}$-Minor-Free or K_5-Minor-Free Graphs*. Algorithmica, vol. 41, pages 245–267, (2004).

[Die97]   R. Diestel. *Graph Theory*. Second Edition. Springer-Verlag, Graduate Texts in Mathematics, vol. 173, (1997).

[DR+03]   J. Cheetham, F. Dehne, A. Rau-Chaplin, U. Stege, and P. J. Taillon. *Solving Large FPT Problems On Coarse Grained Parallel Machines*. Journal on Computer and System Sciences, vol. 67(4), pages 691–706, (2003).

[EK72]   J. Edmonds, and R. M. Karp, *Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems*. Journal of the Association for Computing Machinery, vol. 19, pages 248–264, (1972).

[Fel87]   M. Fellows. *The Robertson-Seymour Theorems: A Survey of Applications*. Contemporary Mathematics, vol. 89, pages 1–18, (1987).

[Fel02]   M. Fellows. *Paramterized Complexity: The Main Ideas and Connections to Practical Computing.* Electronic Notes in Theoretical Computer Science, vol. 61, (2002).

[Fel03]   M. Fellows. *Blow-Ups, Win/Win's, and Crown Rules: Some New Directions in FPT.* Proceedings of 29th Workshop on Graph Theoretic Concepts in Computer Science, Springer Lecture Notes in Computer Science, vol. 2880, pages 1–12, (2003).

[FH+04]   M. Fellows, P. Heggernes, F. Rosamond, C. Sloper, and J.A. Telle. *Exact Algorithms for Finding k Disjoint Triangles in an Arbitrary Graph.* Proceedings of 30th Workshop on Graph Theoretic Concepts in Computer Science (WG '04), Springer Lecture Notes in Computer Science, vol. 3353, pages 235-244, (2004).

[FK02]   S. Fedin, and A. Kulikov. *A $2^{|E|/4}$-time Algorithm for MAX-CUT.* Zapiski nauchnyh seminarov POMI, No.293, pages 129–138, (2002). English translation to appear in Journal of Mathematical Sciences.

[FK+04]   M. Fellows, C. Knauer, N. Nishimura, P. Ragde, F. Rosamond, U. Stege, D. Thilikos, and S. Whitesides. *Faster Fixed-Parameter Tractable Algorithms For Matching and Packing Problems.* Proceedings of the 6th Annual European Symposium on Algorithms (ESA '04), Springer Lecture Notes in Computer Science, vol. 3221, pages 311–322, (2004).

[FKW04]   F. Fomin, D. Kratsch, and G. Woeginger. *Exact (Exponential) Algorithms for the Dominating Set Problem.* Proceedings of 30th Workshop on Graph Theoretic Concepts in Computer Science (WG '04), Springer Lecture Notes in Computer Science, vol. 3353, pages 245-256, (2004).

[FL88]   M. Fellows, and M. Langston. *Nonconstructive Tools for Proving Polynomial-Time Decidability.* Journal of the Association for Computing Machinery, vol. 35(3), pages 727–739, (1988).

[FL92]   M. Fellows, and M. Langston. *On Well-Partial-Order Theory and its Applications to Combinatorial Problems of VLSI Design.* SIAM Journal on Discrete Mathematics, vol. 5, pages 117–126, (1992).

[FM03]   M. Fellows and C. McCartin. *On the Parametric Complexity of Schedules to Minimize Tardy Tasks.* Theoretical Computer Science, vol. 298, pages 317–324, (2003).

[FM+00]   M. Fellows, C. McCartin, F. Rosamond, and U.Stege. *Coordinatized Kernels and Catalytic Reductions: An Improved FPT Algorithm for Max Leaf Spanning Tree and Other Problems*. Proceedings of Foundations of Software Technology and Theoretical Computer Science (FSTTCS '00), Springer Lecture Notes in Computer Science, vol. 1974, pages 240–251, (2000).

[FNU01]   N. Funabiki, S. Nishikawa, and S. Umetani. *A Neural Network Parallel Algorithm for Miminum Maximal Matching Problems*, Information Processing Society of Japan Journal, vol.39, No.03, page 6, (2001).

[FT02]   F. Fomin and D. Thilikos. *A New Upper Bound on the Decomposability of Planar Graphs and Fixed Parameter Algorithms.* Technical report, Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, Barcelona, (2002).

[FT03a]   F. Fomin and D. Thilikos. *Dominating Sets and Local Treewidth.* Proceedings of the 11th Annual European Symposium on Algorithms (ESA '03), Springer Lecture Notes in Computer Science, vol. 2832, pages 221–229, (2003).

[FT03b]   F. Fomin and D. Thilikos. *Dominating Sets in Planar Graphs: Branch-Width and Exponential Speed-Up.* Proceedings of the 14th Annual Association for

Computing Machinery SIAM Symposium on Discrete Algorithms (SODA '03), pages 168–177, (2003).

[G02]   W. Gasarch. *The P?NP Poll*. SIGACT NEWS Complexity Theory Column 36, (2002).

[GD05]   S. Gilmour, and M. Dras. *A Two-Pronged Attack in the Dragon of Intractability*. Proceedings of the 28th Australasian Computer Science Conference, Australian Computer Science Communications, vol. 27(1), pages 183–192, (2005).

[GG+03]   J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. *Graph-Modeled Data Clustering: Fixed-Parameter Algorithms for Clique Generation*. Proceedings of the 5th Italian Conference on Algorithms and Complexity (CIAC '03), Springer Lecture Notes in Computer Science, vol. 2653, pages 108–119, (2003).

[GG+05]   J. Gramm, J. Guo, F. Hüffner, R. Niedermeier, and S. Wernicke. *Improved Fixed-Parameter Algorithms for Two Feedback Set Problems*. Submitted to ACM Symposium on Theory of Computing (STOC '05), (2005).

[GH+03]   J. Gramm, E. Hirsch, R. Niedermeier, and P. Rossmanith. *Worst-Case Upper Bounds for MAX-2-SAT with Application to MAX-CUT*. Discrete Applied Mathematics, vol. 130(2), pages 139–155, (2003).

[GJ79]   M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, San Francisco, (1979).

[GJS76]   M. Garey, D. Johnson, and L. Stockmeyer. *Some Simplified NP-complete Graph Problems*. SIAM Journal on Computing, vol. 1, pages 237–267, (1976).

[GMM94]   G. Galbiati, F. Maffioli, and A. Morzenti. *A Short Note on the Approximability of the Maximum Leaves Spanning Tree Problem.* Information Processing Letters, vol. 52, pages 45–49, (1994).

[GMM97]   G. Galbiati, A. Morzenti and F. Maffioli. *On the Approximability of some Maximum Spanning Tree Problems.* Theoretical Computer Science, vol. 181, pages 107–118, (1997).

[GW92]   J. Grigs, and M. Wu. *Spanning Trees in Graphs with Minimum Degree Three or Four.* Discrete Mathematics, vol. 86, pages 167–183, (1992).

[GW95]   M. Goemans, and D. Williamson. *Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming,* Journal of the Association for Computing Machinery, vol. 42, pages 1115–1145, (1995).

[GY80]   F. Gavril, and M. Yannakakis. *Edge Dominating Sets in Graphs,* SIAM Journal Applied Mathematics, vol. 38, pages 364–372, (1980).

[HHS98]   T. Haynes, S. Hedetniemi, and P. Slater. *Fundamentals of Domination in Graphs.* Marcel Dekker, page 16, (1998).

[HK73]   J. Hopcroft, and R. Karp. *An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs.* SIAM Journal on Computing, vol. 2, pages 225–231, (1973).

[HK78]   P. Hell, and D. Kirkpatrick. *On the Complexity of a Generalized Matching Problem.* Proceedings of 10th Association for Computing Machinery Symposium on Theory of Computing, pages 309–318, (1978).

[HMR01]   B.V. Halldorsson, J.S. Minden, R. Ravi. *PIER: Protein Identification by Epitope Recognition.* Currents in Computational Molecular Biology, pages 109–110, (2001).

[Hol81]   I. Holyer. *The NP-Completeness of Some Edge-partition Problems*. SIAM
            Journal on Computing, vol. 10, pages 713–717, (1981).

[HS89]   C. Hurkens, and A. Schrijver. *On the Size of Systems of Sets Every t of Which
            Have an SDR, with Application to Worst Case Ratio of Heuristics for Pack-
            ing Problems*. SIAM Journal on Discrete Mathematics, vol. 2, pages 68–72,
            (1989).

[IPZ01]   R. Impagliazzo, R. Paturi, and F. Zane. *Which Problems Have Strongly Ex-
            ponential Complexity*. Journal Computer and Systems Sciences, vol. 63(4),
            pages 512–530, (2001).

[Kann94]   V. Kann. *Maximum Bounded H-Matching is* MAX-SNP-*Complete.* Informa-
            tion Processing Letters, vol. 49, pages 309–318, (1994).

[Karp72]   R. Karp. *Reducibility Among Combinatorial Problems.* Complexity of Com-
            puter Computations, Plenum Press, New York, pages 85–103, (1972).

[KK+95]   E. Kranakis, D. Krizanc, B. Ruf, J. Urrutia, and G. Woeginger. *VC-
            Dimensions for Graphs.* Proceedings of Graph Theoretic Concepts in Com-
            puter Science (WG '95), Springer Lecture Notes in Computer Science, vol.
            1017, pages 1–13, (1995).

[KM+99]   S. Khanna, R. Motwani, M. Sudan, and U. Vazirani. *On syntactic versus com-
            putational views of approximability*, SIAM Journal on Computing, vol. 28,
            pages 164–191, (1999).

[KR02]   S. Koth, and V. Raman. *Parameterized Complexity of Finding Subgraphs
            with Hereditary Properties.* Theoretical Computer Science, vol. 289, pages
            997–1008, (2002).

[KR92]   S. Kapoor, and H. Ramesh. *Algorithms for Generating all Spanning Trees of
            Undirected, Directed and Weighted Graphs*. Proceedings of Workshop on Al-

gorithms and Data Structures (WADS '91). Springer Lecture Notes in Computer Science, vol. 519, pages 461–472, (1992)

[KT95]   P. Kolaitis, and M. Thakur. *Approximation Properties of NP Minimization Classes*. Journal Computer System Sciences, vol. 50, pages 391–411, (1995).

[KW91]   D. Kleitman, and D. West. *Spanning Trees with Many Leaves.* SIAM Journal on Discrete Mathematics, vol. 4, pages 99–106, (1991).

[Kur30]   K. Kuratowsky. *Sur le Problème des Courbes Gauches en Topologie*. Fundamenta Mathematicae, vol. 15, pages 271–283, (1930).

[LP86]   L. Lovasz, and M. Plummer. *Matching Theory*. Annals of Discrete Mathematics, vol. 29, North Holland, (1986).

[LR98]   H. Lu, and R. Ravi. *Approximating Maximum Leaf Spanning Trees in Almost Linear Time*. Journal of Algorithms, vol.29, pages 132–141, (1998).

[McC03]   C. McCartin. *Contributions to Parameterized Complexity*. PhD Thesis, Victoria University of Wellington, New Zealand, (2003).

[McS89]   W. McCuaig, and B. Shepherd. *Domination in Graphs with Minimum Degree Two*. Journal of Graph Theory, vol. 13(6), pages 749–762, (1989).

[MPS04]   L. Mathieson, E. Prieto, and P. Shaw. *Packing Edge Disjoint Triangles: A Parameterized View*. Proceedings of International Workshop on Parameterized and Exact Computation (IWPEC '04), Springer Lecture Notes in Computer Science, vol. 3162, pages 127–138, (2004).

[MR99]   M. Mahajan, and V. Raman. *Parameterizing Above Guaranteed Values: MaxSat and MaxCut*. Journal of Algorithms, vol. 31(2), pages 335–354, (1998).

[MV80]   S. Micali, and V. Vazirani. *An $\mathcal{O}(\sqrt{|V|}|E|)$ Algorithm for Finding Maximum Matchings in General Graphs*. Proceedings of IEEE Annual 21st Symposium on Foundations of Computing, pages 17–27, (1980).

[Nie02]   R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*, Habilitation thesis, Universität Tübingen, (2002).

[NR99]   R. Niedermeier, and P. Rossmanith. *Upper Bounds for Vertex Cover Further Improved*. Proceedings of the 16th Symposium on Theoretical Aspects of Computer Science (STACS '99), Springer Lecture Notes in Computer Science, vol. 1563, pages 561–570, (1999).

[NR00]   R. Niedermeier, and P. Rossmanith. *A General Method to Speed Up Fixed-Parameter-Tractable Algorithms*. Information Processing Letters, vol. 73, pages 125–129, (2000).

[NT75]   G. Nemhauser, and L. Trotter. *Vertex Packing: Structural Properties and Algorithms*. Mathematical Programming, vol. 8, pages 232–248, (1975).

[P05]   E. Prieto. *The Method of Extremal Structure on the k-MAXIMUM CUT Problem*. Proceedings of Computing: The Australasian Theory Symposium (CATS '05), Australian Computer Science Communications, vol. 27(4), pages 119–126, (2005).

[PS03]   E. Prieto, and C. Sloper. *Either/Or: Using Vertex Cover Structure in designing FPT-algorithms - the Case of k-Internal Spanning Tree*. Proceedings of Workshop on Algorithms and Data Structures (WADS '03), Springer Lecture Notes in Computer Science vol. 2748, pages 474–483, (2003).

[PS04]   E. Prieto, and C. Sloper. *Looking at the Stars*. Proceedings of International Workshop on Parameterized and Exact Computation (IWPEC '04), Springer Lecture Notes in Computer Science vol. 3162, pages 138–149, (2004).

[PY91]   C. Papadimitriou, and M. Yannakakis. *Optimization, Approximation, and Complexity Classes*. Journal Computing System Sciences, vol. 43, pages 425–440, (1991).

[R96]   B. Reed. *Paths, Stars and the Number Three*. Combinatorics, Probability and Computing, vol. 5, pages 277–295, (1996)

[RBIL02]   R. Rizzi, V. Bafna, S. Istrail, and G. Lancia. *Practical Algorithms and Fixed-Parameter Tractability for the Single Individual SNP Haplotyping Problem*. Proceedings of the 2nd Workshop on Algorithms in Bioinformatics, Springer Lecture Notes in Computer Science, vol. 3240, pages 29–43, (2002).

[Rob86]   J. M. Robson. *Algorithms for Maximum Independent Sets*. Journal of Algorithms, vol. 7, pages 425–440, (1986).

[RS95]   N. Robertson, and P.D. Seymour. *Graph Minors XIII. The Disjoint Paths Problem*. Journal of Combinatorial Theory, Series B, vol. 63(1), pages 65–110, (1995)

[RS99]   N. Robertson, and P.D. Seymour. *Graph Minors XX. Wagner's Conjecture*. to appear.

[RSV04]   B. Reed, K. Smith, and A. Vetta. *Finding Odd Cycle Transversals*. Operations Research Letters, vol. 32, pages 299–301 (2004).

[S98]   R. Solis-Oba. *2-Approximation Algorithm for Finding a Spanning Tree with Maximum Number of Leaves*. Proceedings of the 6th Annual European Symposium on Algorithms (ESA '98), Springer Lecture Notes in Computer Science vol. 1461, pages 441–452 (1998).

[Ski97]   S. Skiena. *The Algorithm Design Manual*. Springer-Verlag, (1997).

[Slo05]   C. Sloper. *Parameterized Algorithms - Crown Reductions and Other Techniques*. PhD Thesis, to appear, (2005).

[Ste00]   U. Stege. *Resolving Conflicts in Problems from Computational Biology*. PhD
           Thesis no. 13364, ETH Zürich, Switzerland, (2000).

[Wag37]   K. Wagner. *Über eine Eigenschaft der ebenen Komplexe*. Mathematische An-
           nalen, vol. 114, pages 570–590, (1937).

[Wei98]   K. Weihe. *Covering Trains by Stations or The Power of Data Reduction.*
           Proceedings of 1st Workshop on Algorithm Engineering and Experiments
           (ALENEX '98), pages 1–8, (1998).

[Wer03]   S. Wernicke. *On the algorithmic Tractability of Single Nucleotide Polymor-
           phism (SNP) Analysis and Related Problems*. Diplomarbeit, Universitat Tub-
           ingen, (2003).

[Woe03]   G. Woeginger. *Exact Algorithms for NP-hard Problems: A Survey*. Combina-
           torial Optimization - Eureka! You shrink!, Springer Lecture Notes in Com-
           puter Science, vol. 2570, pages 185–207, (2003)

[Zit99]   M. Zito. *Randomised Techniques in Combinatorial Algorithmics*. PhD thesis,
           Department of Computer Science, University of Warwick, UK, pages 116–
           126, (1999).