# TECHNIQUES IN PARAMETERIZED ALGORITHM DESIGN

**Christian Sloper**
Cand. Scient.

The University of Bergen
Norway

2005

ISBN 82-308-0108-8
Bergen, Norway 2006

# TECHNIQUES IN PARAMETERIZED ALGORITHM DESIGN

**Christian Sloper**
Cand. Scient.

**Submitted as part of the requirements for**

**Doctor of Philosophy**

Department of Informatics

The University of Bergen
Norway

2005

# ABSTRACT

In this thesis we give a novel classification of techniques for designing parameterized algorithms, together with research publications applying these techniques, in particular Crown Decompositions, to various problems.

In Part I we argue that the currently known techniques can be organized into just four general themes: Bounded Search Trees, Reduction Rules, Induction and Win/Win. The four main themes and their variations are presented through an algorithmic skeleton and illustrated by examples.

Part II contains four research papers that apply the techniques described in Part I on the following problems: MAX INTERNAL SPANNING TREE, $K_3$-PACKING, $H$-PACKING, $K_{1,s}$-PACKING, $P_2$-PACKING, SET SPLITTING, and MAX CUT. The techniques used include Win/Win, Bounded Search Trees, Greedy Localization, Crown Decomposition, Modelled Crown Decomposition, Extremal Method, and Reduction Rules.

# ACKNOWLEDGEMENTS

I would like to express my gratitude to several people for helping me write this thesis.

First and foremost, I thank Professor Jan Arne Telle, my advisor. Without his patient guidance it would not have been possible for me to complete this work. Choosing him as my advisor has proven to be one of the best decisions I have made.

Second, I would like to thank Michael Fellows and Jan Kratochvíl for letting me visit them and their institutions and for supporting me for six months each.

I would also like to thank all my co-authors Marc Bezem, Carlos Cotta, Michael Fellows, Pinar Heggernes, Tore Langholm, Daniel Lokshtanov, Pablo Moscato, Elena Prieto, Frances Rosamond, and Jan Arne Telle.

Then I would like to thank Yngve Villanger for very useful conversation about many parts of this thesis, and Olav Hjortås for his diligence when proofreading this thesis.

My mother, father, and brother deserve thanks for their unwavering support and faith in me, especially in times it was not deserved.

Fortunately, my friends are too many to name, but they have all been instrumental in keeping me in touch with the real world. I have never been able to voice how much you all mean to me.

Finally, I would like to thank one person who in particular transcends all categories above. Being my most prolific co-author, one of my closest friends, an advisor in life if not science, and some have even claimed my Siamese twin separated at birth: Elena, thank you.

# CONTENTS

# 1

# INTRODUCTION

The fundamental realization that not all decision problems have algorithms with efficient running time (unless the unlikely P=NP) was made in the sixties and early seventies [E65, C71, K72]. Since then computer scientists have classified problems as 'good': those that are known to have algorithms with polynomial running time; and 'bad': those that are not.

Unfortunately, many or most of the problems we seek to solve on a day-to-day basis are 'bad'. The need to solve these problems has produced a variety of different ideas on how to cope with them. One of the newest of these ideas is 'Parameterized Complexity'.

At the core of parameterized complexity lies the realization that most problems, especially in a practical setting, are not so generic as the abstract problem description we find in problem collections like Garey and Johnson [GJ79]. We could for example know something about the input instances (e.g., graphs could have bounded treewidth or restricted genus) or we might require something about the output solution (e.g., a solution may only be interesting if it is not too large). Extra information like this is often possible to quantify numerically and leads to the notion of a parameter.

In classical complexity the NP-complete problems are indistinguishable from each other in terms of hardness. The introduction of a parameter changes this. We can show that a problem that is otherwise intractable has an efficient algorithm as long as the parameter is kept small. For other problems the introduction of a parameter does little to improve the situation, and the problem remains intractable. This division of the classically inseparable NP-complete problems is in itself interesting, and much work has been done on the complexity theoretic aspect [DEFPR03, DF99, FG01, FG02, G99]. This is, however, not the topic of this thesis. We will instead focus on the problems that do become tractable in fixed parameter complexity and in particular on the known techniques for designing algorithms for these problems.

As the field has developed in the last decade, various ideas have been put forward on how to develop efficient parameterized algorithms. Since this field is quite young, the contributing authors have been very free to devise new techniques and thus also name their ideas. This has resulted in a myriad of names, and when techniques have again been

renamed by survey papers, we are left with a confusing tangle that can be difficult to penetrate for new readers.

Aside from several survey articles there have been at least two comprehensive texts written on parameterized algorithm design: Downey and Fellows's seminal book 'Parameterized Complexity' [DF99] and the habilitation thesis of Rolf Niedermeier, 'Invitation to Fixed-Parameter Algorithms' [N02]. Downey and Fellows's book came very early in the history of parameterized complexity, and thus does not include the newer ideas. Moreover, it does not spend much time on classifying various techniques. Niedermeier's habilitation thesis is dedicated to algorithmic techniques. He singles out Bounded Search Trees and Kernelization as the two major techniques, and spends considerable time giving a detailed explanation of both. Under the generic heading 'Further Algorithmic Techniques', Niedermeier lists Integer Linear Programming, Color Coding, Dynamic Programming and Tree Decompositions.

Some of the techniques given a name in the literature include Bounded Search Tree [DF99], Search Trees [N02], Data Reduction [N02], Kernelization [DF99] The Extremal Method [FM+00], The Algorithmic Method [P05], Catalytic Vertices [FM+00], Crown Reductions [CFJ04], Modelled Crown Reductions [DFRS04], Either/Or [PS03], Reduction to Independent Set Structure [PS04], Greedy Localization [DFRS04], Win/Win [F03], Iterative Compression [DFRS04], Well-Quasi-Ordering [DF99], FPT through Treewidth [DF99], Integer Linear Programming [N02], Color Coding [AYZ95], Method of Testsets [DF99], and Interleaving [NR00].

According to this list, there are over twenty differently named techniques that could be applied when we want to construct a fixed parameter algorithm for a problem. We feel that this gives a wrong picture of parameterized algorithm design.

The number of distinct design techniques is in our opinion considerably fewer. Many of the techniques build on the same general idea, and as a problem solver it is important to be familiar with these general themes and ideas. The details will in any case vary from problem to problem.

In this thesis we have attempted to present and categorize these general themes under the four main headings: Bounded Search Trees, Reduction Rules, Induction, and Win/Win. We have selected what we consider to be the most important and general techniques, and consider the remainder to be either variations or combinations of the themes presented in the first six chapters of this thesis. An illustration of the techniques we have chosen to present, and the categorization we have chosen, can be seen in Figure 1.1. Let us briefly mention techniques in the above list that are not in Figure 1.1. Data Reduction, Kernelization are different names for Reduction Rules and Either/Or is an earlier name for Win/Win. Catalytic Vertices is a variant of Reduction Rules, while Interleaving is a combination of the techniques Bounded Search Tree and Reduction Rules. Integer Linear Programming, which is based on a result by Lenstra [L83], can be viewed as a Win/Win

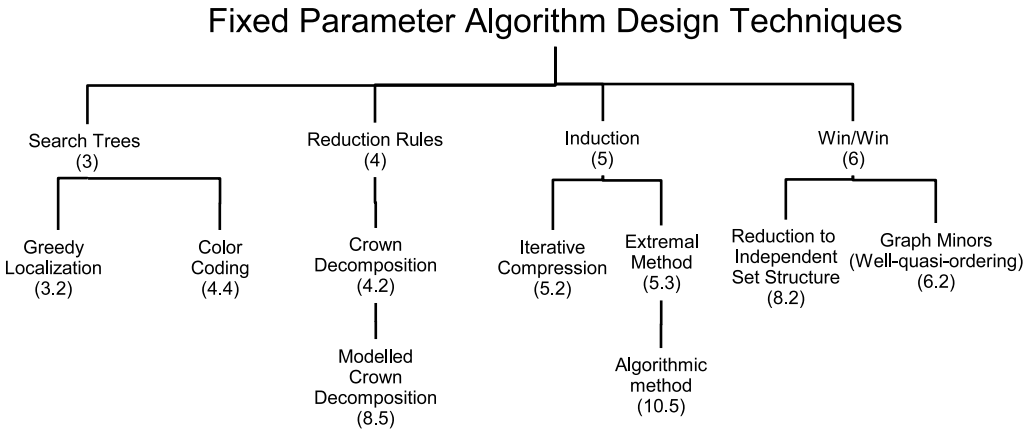


Figure 1.1: Our classification of parameterized algorithm techniques, labelled by chapter number.

algorithm.

The thesis is in two parts, with the second part containing the main research contribution in terms of published papers. The first part consists of chapters 2 through 6 and is an overview of the various parameterized algorithm design techniques. For each technique, we present a basic introduction, a few examples and the most important variations of the main technique.

The examples we have included have been selected on the criteria that they should be as simple as possible. This is because we want to focus on the core idea of the technique. Thus, we have not been afraid to use well known examples from the literature whenever we felt that these examples were the best to show the relevant technique.

For each basic technique, we provide an 'algorithm skeleton' in pseudo-code giving a rough outline of the algorithmic idea behind the technique. For most of the examples we provide a pseudo-code algorithm, describing the algorithm as well as proving its correctness.

When giving pseudo-code there is always a question of how close to 'real code' it should be. If the pseudo-code given is very close to a computer program, it is easier to translate to real code and implement, but on the other hand it becomes more detailed and we lose some flexibility which in turn lowers readability. We have tried to find a balance between real code and readability, by giving pseudo-code in the style of an iterative program (e.g. c++ or Java), but refraining from using too much detail. The algorithms are explained in more detail in the text when needed.

## 1.1 OVERVIEW

The individual chapters are organized as follows:

Chapter 2 outlines the notation and basic definitions used in Part I.

Chapter 3 gives an overview of techniques using search trees to find optimal solutions. Basic bounded search trees focus on limiting the fan-out and the height of a search tree. We need to limit the height to a function of $k$, and we show that it is sufficient to limit the fan-out to $\mathcal{O}((\log n)^{g(k)} \cdot f(k))$ for arbitrary functions $f$ and $g$. In addition to the basic search tree idea we have included Greedy Localization where we make use of a greedy algorithm to obtain initial information that we then complete using a search tree. Finally we describe the related Color Coding technique, that branches into a set of instances where vertices have been pre-colored using $k$ colors.

Chapter 4 describes the techniques applying reduction to shrink the problem to a more manageable size. This is often done through reduction rules, where we identify a local structure in a graph. By modifying or removing this structure, we can reduce the size of the graph or the parameter without introducing or removing solutions. Aside from the basic reduction rule algorithms we cover the more advanced use of crown reduction rules. We explain Crown reductions in detail as all of the case studies in Part II use crown decompositions to some degree. In addition to an example of an algorithm using a crown reduction rule, we include a section on its complexity where we prove that finding a crown decomposition of a given size is NP-complete.

In Chapter 5 we show how induction can be used as a base for creating parameterized algorithms. We show how a certificate for smaller instances can be updated to a certificate for a larger instance. We discuss some of the requirements that is needed for an inductive algorithm to be applicable and then describe two techniques that use induction at their cores. For minimization we present Iterative Compression with several examples, and for maximization we show that the natural inductive algorithm is equivalent to the algorithmic version of the Extremal Method. We also present the Extremal Method in this chapter.

Chapter 6 gives a design technique that ties a problem $A$ to a problem $B$ in such a way that both a 'Yes' and a 'No' answer for $B$ can help us solve $A$. Since both cases are helpful when designing algorithms for $A$ this technique is dubbed 'Win/Win'. We showcase this very strong technique by tying PLANAR DOMINATING SET to BRANCHWIDTH and use this relationship to give an elegant $\mathcal{O}^*(c^{\sqrt{k}})$ algorithm for PLANAR DOMINATING SET. Here we also present the Graph Minor theorem, and show that it can be used to give parameterized algorithms for graph problems whenever the 'Yes'-instances or the 'No'-instances are closed under minors.

Chapter 7 is a list of the problems used in the first part of the thesis. We have included this chapter as some of the problems are used more than once. Thus, in the text we will

list a problem with small caps and then give a reference to Chapter 7 where the complete problem definition can be found. E.g., PROBLEM (P7.15) is problem 15 of Chapter 7. There is a bibliography for Part I at the end of this chapter.

Part II of the thesis is based on four published papers. Each chapter discusses one or more parameterized problems, and gives algorithms and kernels for these using the techniques described in part I.

Chapters 8, 9, 10, and 11 are copies of four published papers with merely typographic changes.

Chapter 8 is a paper co-authored with Elena Prieto. It is based on the paper 'Either/Or: Using Vertex Cover Structure in designing FPT-algorithms - the case of $k$-Internal Spanning Tree' [PS03] that originally appeared at the 'Workshop on Algorithms and Datastructures' in Ottawa, 2003. Some of the results were improved and a new version, 'Reducing to Independent Set Structure — the Case of $k$-INTERNAL SPANNING TREE' has been accepted to the 'Nordic Journal of Computing' [PS05]. In this paper, we look at the problem of constructing a spanning tree with many internal nodes. This paper uses a Win/Win strategy together with a crown decomposition, obtaining a quadratic kernel for the problem.

Chapter 9 is a paper co-authored with Elena Prieto titled 'Looking at the Stars' [PS04]. This paper originally appeared at 'First International Workshop on Parameterized and Exact Computation' in Bergen, 2004. It has later been accepted for a special issue of 'Journal of Theoretical Computer Science'. Here we give a quadratic kernel for finding vertex disjoint copies of $K_{1,s}$ for any $s$, using the extremal method. We give a linear kernel for finding vertex disjoint copies of $P_2$ using a crown reduction algorithm.

Chapter 10 is a paper co-authored with Michael Fellows, Pinar Heggernes, Frances Rosamond and Jan Arne Telle titled 'Finding $k$ disjoint triangles in an arbitrary graph' [FHRST04]. This paper appeared at the conference 'Workshop on Graph-Theoretic Concepts in Computer Science' in Bonn, 2004. In this paper we discuss packing problems in general and describe an algorithm for finding vertex disjoint copies of $K_3$ in graphs. We give a cubic kernel and a $\mathcal{O}^*(2^{\mathcal{O}(k \log k)})$ algorithm for this problem using the technique Modelled Crown Decompositions. Later this result has been improved [FKNRSTW04], giving a better running time of $\mathcal{O}^*(2^{\mathcal{O}(k)})$.

Chapter 11 is the paper 'Fixed Parameter Set Splitting, Linear Kernel and Improved Running Time' [LS05] co-authored with Daniel Lokshtanov. It appeared at the conference 'Algorithms and Complexity in Durham 2005'. Here we study two-colorings of hypergraphs and give a linear kernel for the problem and also an efficient algorithm based on using Win/Win and crown decomposition techniques.

# 2

# NOTATION AND DEFINITIONS

## 2.1 COMPUTATIONAL MODEL

We will assume a single processor, *random-access machine* as the underlying machine model throughout this thesis. In the random-access machine any simple operation (arithmetic, if-statements, memory-access etc.) takes unit length of time. The word size is sufficiently large to hold any number in our algorithms. We will not exploit this by for instance encoding more than one natural number in each word.

## 2.2 SETS

A *set* is a finite or infinite collection of objects in which order has no significance, and no object appear more than once. We will refer to the members of a set as *elements* and the notation $e \in A$ is used to denote that is $e$ an element of a set $A$. Similarly we use $e \notin A$ to denote that $e$ is not an element in $A$.

We will use the set of natural numbers $\mathbb{N}= \{1, 2, 3, \ldots\}$.

A partition of a set $X$ is a set of nonempty subsets of $X$ such that every element $x$ in $X$ is in exactly one of these subsets.

## 2.3 RELATIONS AND FUNCTIONS

A *binary relation* $R$ on two sets $A$ and $B$ is a subset of the Cartesian product $A \times B$. Given two sets $A$ and $B$, a *function* $f$ is a binary relation on $A \times B$ such that for all $a \in A$ there exists precisely one $b \in B$ such that $(a, b) \in f$. We will use the notation $f : A \to B$ to describe a function $f$ from $A$ to $B$.

## 2.4 BASIC GRAPH THEORY

We assume simple, undirected, connected graphs $G = (V, E)$, where $|V| = n$ and $|E| = m$. The neighbors of a vertex $v$ are denoted by $N(v)$, the closed neighborhood $N[v] =$

$N(v) \cup \{v\}$. For a set of vertices $A \subseteq V$, we have $N(A) = \{v \notin A \mid uv \in E \text{ and } u \in A\}$. The subgraph of $G$ induced by $A$ is denoted by $G[A]$. For ease of notation, we will use informal expressions like $G - u$ to denote $G[V \setminus \{u\}]$, $G - U$ to denote $G[V \setminus U]$, and $G - e$ to denote $G(V, E \setminus \{e\})$, where $u$ is a vertex, $U$ is a vertex set, and $e$ is an edge in $G$.

We say that a *matching* $M$ on a graph $G$ is a set of edges of G such that no two of them have a vertex in common. The largest possible matching on a graph with $n$ nodes consists of $n/2$ edges, and such a matching is called a *perfect matching*. We write $V(M)$ to indicate the set of vertices incident to the edges in $M$.

A tree is a connected graph without cycles. A disconnected graph without cycles is a forest. A *subtree* of a graph $G$ is a subgraph of $G$ that forms a tree. If a subtree contains $n - 1$ edges, it is a *spanning tree* of $G$.

We say that $K_{1,s}$ is a $s$-star or a star of size $s$. The symbol $P_i$ denotes a path of $i + 1$ vertices and $i$ edges.

An *$H$-packing* $W$ of $G$ is a collection of vertex disjoint subgraphs of $G$ each isomorphic to $H$. We will use $V(W)$ to denote the vertices of $G$ that appear in $W$, and $E(W)$ to denote the edges.

The *contraction* of an edge of a graph, also called *edge contraction*, is the graph obtained by replacing the two nodes $v_1, v_2$ with a single node $v_3$ such that $v_3$ is adjacent to the union of the nodes to which $v_1$ and $v_2$ were originally adjacent.

## 2.5 TREE DECOMPOSITIONS AND BRANCH DECOMPOSITIONS

Throughout the text and in particular in Chapter 6 we will make use of tree decompositions of graphs. Tree decompositions tries to give a measure of how tree-like a graph is.

**Definition 2.5.1** *Let $G = (V, E)$ be a graph. A* tree decomposition *of $G$ is a pair $\mathcal{X} = \langle \{X_i \mid i \in I\}, T \rangle$, where each $X_i$ is a subset of $V$, called a* bag, *and $T$ is a tree with the elements of $I$ as nodes. The following three properties must hold:*

1. $\cup_{i \in I} X_i = V$;
2. *for every edge $(u, v) \in E$, there is an $i \in I$ such that $\{u, v\} \subseteq X_i$;*
3. *for $i, j, k \in I$, if $j$ lies on the path between $i$ and $k$ in $T$, then $X_i \cap X_k \subseteq X_j$.*

*The* width $tw(\mathcal{X})$ *of* $\mathcal{X}$ *equals* $\max\{|X_i| \mid i \in I\} - 1$. *The* treewidth $tw(G)$ *of* $G$ *is the minimum* $k$ *such that* $G$ *has a tree decomposition of width* $k$.

We will also use branch decompositions.

**Definition 2.5.2** *A* branch decomposition *of a graph G is a pair* $(T; \mu)$, *where* $T$ *is a tree with vertices of degree one or three and* $\mu$ *is a bijection from the set of leaves* $L$ *of* $T$ *to* $E(G)$. *Let* $e$ *be an edge of* $T$. *The removal of* $e$ *results in two subtrees of* $T$, *say* $T_1$ *and* $T_2$. *Let* $G_i$ *be the graph formed by the edge set* $\{\mu(f) \mid f \in L \cap V(Ti)\}$ *for* $i \in \{1, 2\}$. *The middle set* $mid(e)$ *of* $e$ *is the intersection of the vertex sets of* $G_1$ *and* $G_2$, *i.e.,* $mid(e) := V(G_1) \cap V(G_2)$. *The width of* $(T; \mu)$ *is the maximum size of the middle sets over all edges of* $T$, *and the branch-width of* $G$, $bw(G)$, *is the minimum width over all branch decompositions of* $G$.

## 2.6 $\mathcal{O}$- AND $\mathcal{O}^*$-NOTATION

It is commonplace to express the running time of algorithms with $\mathcal{O}$-notation.

**Definition 2.6.1** *For a function* $f : \mathbb{N} \to \mathbb{N}$, *we write* $f(n) = \mathcal{O}(g(n))$ *if there exists constants* $c$ *and* $n_0$ *such that* $f(n) \leq cg(n)$ *for all* $n \geq n_0$.

Since we throughout this thesis are talking about algorithms with exponential running time, we will adopt the $\mathcal{O}^*$ notation as seen in [W03], which suppresses the polynomials in the running time and focus on the exponentials.

**Definition 2.6.2** *Given a function* $f(n, k)$ *we write* $f(n, k) = \mathcal{O}^*(g(k))$ *if there exist a* $k_0, c$, *and* $n_0$ *such that* $f(n, k) \leq g(k)n^c$ *for all* $k \geq k_0$ *and* $n \geq n_0$

Thus the polynomial part of all terms are left out of the expression when using $\mathcal{O}^*$-notation.

## 2.7 COMPUTATIONAL COMPLEXITY

**Definition 2.7.1** *Let* $A$ *be an algorithm. The* **running time** *or* **time complexity** *of* $A$ *is the function* $f : \mathbb{N} \to \mathbb{N}$, *where* $f(n)$ *is the maximum number of steps that* $A$ *uses on any input of length* $n$. *We also say that* $A$ *runs in time* $f(n)$ *and that* $A$ *is an* $f(n)$ *time algorithm.*

A system of *time complexity classes* have been devised to classify problems according to their time complexity. Each of these classes contain the problems that are asymptotically equivalent in running time.

**Definition 2.7.2** *Time complexity class*

*Let* $t : N \longrightarrow N$ *be a function. The* time complexity class *TIME*$(t(n))$*, is TIME*$(t(n))$ =*{*$L$ | $L$ *is a language decided by an* $\mathcal{O}(t(n))$ *algorithm.*

The class P is the class of problems that are solvable in polynomial time. We consider this class to be roughly equivalent to the class of problems that can be considered computationally easy to solve.

**Definition 2.7.3** P = $\bigcup_{k\in\mathbb{N}} TIME(n^k)$

The class NP is another major class of decision problems in complexity theory. In NP we find all the problems that we can verify with polynomial algorithms. That is, any 'Yes'-instance for a problem in NP has a certificate that we can check in polynomial time.

**Definition 2.7.4** NP $= \{L \mid \exists V$ *such that* $V$ *is a verifier for* $L$ *and* $V \in P\}$

**Definition 2.7.5** *A language* $B$ *is NP-complete if it satisfies the following: (1)* $B$ *is in NP, and (2) every language* $A$ *in NP is polynomial time reducible to* $B$*. If a problem satisfies requirement* $2$*, we say that it is NP-hard.*

Although it is unknown if P = NP, it is widely believed that this is not the case. For the remainder of this thesis we will always operate under the assumption that P$\neq$NP.

## 2.8 FIXED PARAMETER COMPLEXITY

We first define our notion of a parameterized problem, using the definitions given in [N02].

**Definition 2.8.1** *A* parameterized problem *is a language* $L \subseteq \Sigma^* \times \Sigma^*$ *where* $\Sigma$ *is a finite alphabet. The second component is called the* parameter *of the problem.*

**Definition 2.8.2** *A parameterized problem* $L$ *is* fixed-parameter tractable *if the question '*$(x_1, x_2) \in L$*?' can be decided in running time* $f(|x_2|) \cdot |x_1|^{\mathcal{O}(1)}$*, where* $f$ *is an arbitrary function on nonnegative integers. The corresponding complexity class is called FPT.*

It is commonplace to only consider the languages $L \subseteq \Sigma^* \times \mathbb{N}$ as the set of parameterized problems since almost all problems have a non-negative integer as parameter. For all problems in this thesis, the parameter is an integer, so when it is needed the next smaller values of a parameter is well defined.

It is not believed that all NP-complete problems are Fixed Parameter Tractable, the class is split into a hierarchy of classes FPT⊆W[1]⊆W[2]⊆ $\cdots$ ⊆W[P]. Here the classes W[1]⊆W[2]⊆ $\cdots$ ⊆W[P] are intractable, and we justify this by a completeness-result not unlike classical complexity. It is proven in [CCDF97] that $k$-SHORT NONDETERMINISTIC TURING MACHINE ACCEPTANCE (P7.17 is W[1]-complete thus giving strong natural evidence that $FPT \neq W[1]$.

Further background on parameterized complexity can be found in [DF99].

## 2.9 EXACT ALGORITHMS

Exact algorithms is a related field where we calculate the optimal solution for the problem in question, spending exponential time if required. In exact algorithms we consider the NP-hard maximization/minimization versions of the decision problems that Fixed Parameter Complexity tries to deal with. While we accept exponential running time in our algorithms, the goal is to lower the exponential part of the function as much as possible.

Techniques from exact algorithms can often be used in parameterized algorithms and, more importantly, the algorithms can often be combined with linear kernels to achieve strong results for parameterized problems.

An excellent survey paper that can serve as an introduction to this area is written by Woeginger [W03].

# Part I

# An overview of parameterized algorithm design techniques

# 3

# BOUNDED SEARCH TREES

In this chapter we will discuss the most common technique in parameterized algorithm design, Bounded Search Trees. Here we seek to limit the computational load by making an intelligent search for the optimal solution without searching the whole solution space.

We will first discuss the general idea in the bounded search tree technique, give a definition of a bounded search tree algorithm, and give some simple examples. We then discuss the variant Greedy Localization, where we use a greedy algorithm to obtain some useful initial information, and Color Coding, where we use a result from hashing theory to color the vertices.

## 3.1 BASIC SEARCH TREES - CONTROLLED RECURSION

All combinatorial problems have a finite solution space, and a brute-force way to find an optimal solution is to systematically search the entire space. In the Bounded Search Tree technique we search only a part of this space. This is usually done in a tree-like fashion, at each node deciding some part of the input, and creating one branch for each possible choice. At each new branch we are left with an instance with additional information about the solution. For parameterized problems this additional information can take several forms, but often translates into a smaller parameter value and usually also a decrease in the main input size.

Care must be taken to avoid the search tree becoming too large. To prove a problem to be FPT we need to show that the height of the search tree is bounded by a function depending only on $k$, and that the fan-out of the tree, i.e., the maximum number of children of a node, is $\mathcal{O}((\log n)^{g(k)} f(k))$ (see Theorem 3.1.1).

The height of the tree represents the maximum number of choices we are allowed to make before we must be able to determine a solution. For subset problems (problems where we must select a subset $V' \subseteq V(G)$ with some property) we usually regulate the height of the tree by choosing one or more vertices for the solution at each level, thus decreasing our parameter $k$. Since the problem usually is trivial when $k = 0$, we have an upper bound on the height of the tree.

However, it is not sufficient to control only the height. The vertex subset problems have trivial $n^k$ search trees by having nodes with fan-out $n$. In addition to giving a bound on the height it is necessary to show that the maximum fan-out of the tree is $\mathcal{O}((\log n)^{g(k)} f(k))$. We control the fan-out by carefully selecting the criteria to branch on, described in a branching rule. A branching rule identifies a certain structure in the graph (the left hand side LHS) and creates $c \in \mathcal{O}((\log n)^{g(k)} f(k))$ new instances $(G_1, k_1), (G_2, k_2), \ldots, (G_c, k_c)$ such that the instance $(G, k)$ is a 'Yes'-instance if and only if at least one instance $(G_i, k_i)$ is a 'Yes'-instance.

Taking the above into consideration, we say that a bounded search tree algorithm is an algorithm that:

- is recursive, and
- terminates with a polynomial time base case, and
- makes $\mathcal{O}((\log n)^{g(k)} f(k))$ different recursive calls in each recursive step, and
- reaches a polynomial time base case in at most $h(k)$ nested recursive calls.

---

**Algorithm** BST with a set $B$ of branching rules $B_i : LHS_i \rightarrow (G_1, k_1), \ldots, (G_{c_i}, k_{c_i})$
**Input** Graph $G$, integer $k$

**if** $G$ contains some $LHS_i$ **then** Call BST on instances $(G_1, k_1), \ldots, (G_{c_i}, k_{c_i})$
**else** Use polynomial algorithm to solve $(G, k)$, if 'Yes' then output 'Yes' and **halt**

Answer 'No'

---

Figure 3.1: An algorithm skeleton for Bounded Search Tree.

The running time of an algorithm that uses search trees is bounded by the number of nodes of the search tree times the running time for each node. We assume polynomial running time at each node in the search tree, thus only the size of the search tree contributes in $\mathcal{O}^*$-notation. It is normal that an algorithm checks for several different branching rules and branches according to one of the cases that applies. Although search tree algorithms can have many branching rules it is usually straightforward to calculate the search tree's worst case size. For the functions given above, an upper bound on the search tree is $\mathcal{O}(((\log n)^{g(k)} f(k))^{h(k)})$. To show that this is an FPT-function we prove the following:

**Theorem 3.1.1** *There exists a function $f(k)$ such that $(\log n)^k \leq f(k)n$ for all $n$ and $k$.*

**Proof.** First recall that a logarithmic function grows slower than any root-function. That is,

$$\log n = o(n^{1/k}) \quad \forall k.$$

Which implies that

$$\forall k \geq 1 \; \exists n_0 \geq 1 \text{ such that } \forall n \geq n_0 \quad \log n < n^{1/k}$$

Thus there must be a function $h(k) : \mathbb{N} \to \mathbb{N}^+$ such that $\forall k \; \forall n \geq h(k) \quad \log n < n^{1/k}$ and then $\forall k \; \forall n \geq h(k) \quad (\log n)^k < n$.

So now the situation is that for all $n > h(k)$ we have that $(\log n)^k < n$ and the theorem holds. If on the other hand $n \leq h(k)$, then $\log n \leq \log h(k)$ and $(\log n)^k \leq (\log h(k))^k$. Yielding

$$(\log n)^k \leq \max\{1, (\log h(k))^k\} \cdot n$$

□

We gave this proof showing that $(\log n)^k$ is an FPT-function, but it is easy to extend this to show that $\mathcal{O}((\log n)^{g(k)})$ is also an $FPT$-function.

As discussed in section 3.1.2, it is often possible to obtain better upper bounds by a more careful analysis of the branching in the tree.

### 3.1.1 EXAMPLES OF BOUNDED SEARCH TREES

We present a few examples of bounded search tree algorithms, in particular an $\mathcal{O}^*(6^k)$ algorithm for PLANAR INDEPENDENT SET (P7.15) and an $\mathcal{O}^*(1.466^k)$ algorithm for VERTEX COVER (P7.21).

**Theorem 3.1.2** PLANAR INDEPENDENT SET *can be solved in time* $\mathcal{O}^*(6^k)$.

**Proof.** In an instance $(G, k)$ of PLANAR INDEPENDENT SET we know that for any maximal independent set $S \subseteq V(G)$ and for any $v \in V(G)$ it is true that $N[v] \cap S \neq \emptyset$. If this was not the case, we could include $v$ to obtain a larger set, contradicting maximality. This together with the well known fact that any planar graph contains a vertex of degree at most $5$ (this follows trivially from Euler's formula), allows us to continually branch on a low degree vertex $v$ selecting either $v$ or one of at most five neighbors. Selecting a vertex for the independent set lets us remove its closed neighborhood $N[v]$ from the graph. This leaves us with at most six smaller instances, each with parameter $k' = k - 1$.

As the problem is trivial when the parameter is 0, the size of the tree $T(k)$ is bounded by $6 \cdot T(k-1)$. This recurrence relation becomes $T(k) \leq 6^k$. □

**Theorem 3.1.3** VERTEX COVER *can be solved in time* $\mathcal{O}^*(1.466^k)$.

**Proof.** To prove this we make two initial observations. First, that any valid vertex cover must contain either a vertex $v$ or all its neighbors $N(v)$. Second, if a graph has only vertices of degree less than three, then VERTEX COVER is polynomial time solvable. The second observation gives us a clear goal: we will try to reach a polynomial time instance by branching on any vertex of degree higher than two. We can do this as each high degree vertex $v$ creates only two branches, where one branch decreases the parameter by one and the other decreases the parameter by $|N(v)|$ (three or more). The recursive function $T(k) \leq T(k-1)+T(k-3)$ gives a bound on the size of the tree. This recursive function can be solved by finding the largest root of its characteristic polynomial $\lambda^k = \lambda^{k-1}+\lambda^{k-3}$. Using standard computer tools, this root can be estimated to 1.466, giving the desired result.

□

We can see that both examples follow the outline given for search tree algorithms. They are recursive, have a constant number of recursive calls in each iteration and reach a polynomial base case in at most $k$ nested calls.

### 3.1.2 HOW IS IT USED IN PRACTICE?

Bounded Search Trees is one of the most common and successful parameterized algorithm design techniques in the field today and very many examples exist. The very powerful VERTEX COVER (P7.21) algorithm by Chen, Kanj and Jia [CKJ01] achieving a running time of $\mathcal{O}(1.286^k)$ is a bounded search tree algorithm and remains one of the most cited parameterized algorithms. The main strength, some would claim also the main drawback, of these algorithms is that they can often be continually refined by considering ever more complicated cases. This allows for repeated improvement in theoretical performance, but at the cost of elegance and clarity. There are examples where the algorithm reaches enormous size. For example, the exact algorithm for Maximum Independent Set by Robson [R01] has tens of thousands computer generated cases. Too many cases will also hurt the practicality of implementing and running such algorithms, as each case causes unavoidable overhead. Some implementations of the vertex cover algorithm deliberately skip some of the more advanced cases as they in most practical instances only slow the algorithm down, as reported in [L04].

## 3.2 GREEDY LOCALIZATION

In this section we present a variation of the bounded search tree technique that uses a clever first branching to start off the recursive search for the solution. We start with a greedy algorithm to construct a solution. If the problem is of the correct size, then we are done, otherwise we argue that any optimal solution must intersect with the greedy solution in a certain way, giving us additional information which we use to branch.

A simple example for this is $K_3$-PACKING (P7.9). Note that a faster algorithm for this problem using a different technique can be found in Chapter 10.

**Theorem 3.2.1** *$K_3$-Packing can be solved in $\mathcal{O}^*(\binom{3k}{k} \cdot (2k)!)$ time.*

**Proof.** Note that we can obtain a maximal $K_3$-packing $C$ using a simple greedy algorithm. If this packing is of size $k$ or larger we are done, if not we observe that every $K_3$ in any optimal packing $Opt$ must intersect $C$ in at least one vertex (otherwise there would be a $K_3$ in $G - C$ not included by the greedy algorithm).

Since our greedy set has less than $3k$ vertices, we can now search for $k$ vertices in the intersection $V(C) \cap V(Opt)$. There are at most $\binom{3k}{k}$ ways to choose these $k$ vertices and we create one branch for each of these possibilities.

In each branch we now have $k$ partial sets, each containing one vertex. Our goal is to complete such a collection of partial sets vertex by vertex. In each tree-node we will again use a greedy tactic, by completing each of the partial sets to a $K_3$ in a greedy manner. If we succeed to complete all $k$ sets $S_1, S_2, \ldots, S_k$ in a branch, we can safely halt and answer yes, but most likely we will fail in completing some set $S_j$. Assuming there is a way to correctly complete the partial sets $S_1, \ldots, S_k$ into $k$ disjoint copies of $K_3$, the only reason we could fail to complete $S_j$ is because one of the earlier vertices we had selected to be added to $S_1, S_2, \ldots, S_{j-1}, S_{j+1}, \ldots, S_k$ should instead be in $S_j$. Thus we have identified a set $Q$ of at most $2k$ vertices of which at least one should belong to $S_j$. Only at this point do we branch again, with fan-out equal to $|Q|$, creating one branch for each possible addition to $S_j$. We have in this way added one more vertex to our partial set, and we can repeat the process on each new level. Since we start with $k$ vertices, add one vertex at each level and need $3k$ vertices in total, the height of the tree is at most $2k$.

The total size of such a tree is at most $\binom{3k}{k} \cdot (2k)!$. Here, $\binom{3k}{k}$ is a result of the initial branching, and $(2k)!$ is a factor since each set $Q$ which we branch on has its size at any level $h$ bounded by $2k - h$. □

This is a bounded search tree algorithm, as we now argue. The algorithm is clearly recursive and since we branch on the greedy solution we satisfy the requirement concerning the fan-out, $\binom{3k}{k}$ recursive calls in the first step, and then at most $2k$ in the following

steps. Furthermore we terminate after at most $2k$ nested calls with a trivially polynomial instance.

## 3.3 COLOR CODING

Color Coding is a technique that was introduced by Alon, Yuster, and Zwick in their paper 'Color Coding' [AYZ95]. Given an input to a parameterized graph problem we color the vertices with $k$ colors such that the structure we are looking for will interact with the color classes in a specific way. To do this we create many branches of colored graphs, using a family of perfect hash functions for the coloring.

**Definition 3.3.1** *A $k$-perfect family of hash functions is a family $\mathcal{H}$ of functions from $\{1,\ldots,n\}$ onto $\{1,\ldots,k\}$ such that for each $S \subset \{1,\ldots,n\}$ with $|S| = k$ there exists an $h \in \mathcal{H}$ that is bijective when restricted to $S$.*

Schmidt and Siegal [SS90] describe a construction of a $k$-perfect family of hash functions of size $2^{\mathcal{O}(k)} \log^2 n$, and [AYZ95] describes how to obtain an even smaller one of size $2^{\mathcal{O}(k)} \log n$.

The technique applies a family of perfect hash functions to partition vertices of the input graph into $k$ sets, i.e., $k$ different colors. By the property of perfect hash families we know that for any $k$-sized subset $S$ of the vertices, one of the hash functions in the family will color each vertex in $S$ with a different color. Thus, if we seek a $k$-set $C$ with a specific property (e.g., a $k$-cycle), we know that if there is such a set $C$ in the graph then its vertices will, for at least one function in the hash family, be colored with each of the $k$ colors. See the algorithm skeleton in Figure 3.2.

A major drawback of these algorithms is that while the hash family has an asymptotically good size, the $\mathcal{O}$-notation hides a large constant. Thus, from a practical viewpoint the color coding algorithms would be slower than, for example, a $2^{k \log k}$ algorithm.

We argue that Color Coding is closer to Bounded Search Tree algorithms than any other class of FPT algorithms. Although the algorithm skeleton in Figure 3.2 is recursive only in the strictest terms (it calls itself 0 times), it satisfies the other three requirements much better. The fan out is bounded by $2^{\mathcal{O}(k)} \log n$ which is a FPT function by Lemma 3.1.1, and the height of our tree (the number of nested calls) is only 1. The final requirement is that the algorithm should terminate in a polynomial time base case. In Color Coding it depends on the problem we are solving whether or not these base cases are polynomial. Thus we feel it is correct to consider Color Coding an application of Bounded Search Trees.

**Algorithm** Skeleton structure for Color Coding
**Input:** A graph $G = (V, E)$ and an integer $k$.
**Output:** A 'Yes' or a 'No' answer

Let $\mathcal{F}$ be a family of perfect hash functions from $\{1, \ldots, n\}$ to $\{1, \ldots, k\}$ where $|V| = n$.

**For** each function $f \in \mathcal{F}$
    Use $f$ to color $V$ using $k$ colors.
    **run** algorithm for the colored problem
    **if** 'Yes' **then** Answer 'Yes' and **halt**
**End For**
Output a 'No' Answer.

Figure 3.2: A skeleton structure for 'Color Coding'

### 3.3.1 EXAMPLE OF A COLOR CODING ALGORITHM

To give a simple example of how to use this we give an algorithm for the $k$-CYCLE (P7.2) problem, which asks for a cycle of length exactly $k$. This problem is obviously NP-complete since it is equivalent to HAMILTONIAN CYCLE(7.6) for $k = n$. Let us consider algorithm '$k$-cycle algorithm' in Figure 3.3.

**Theorem 3.3.1** *The '$k$-cycle algorithm' is correct and runs in time $\mathcal{O}^*(2^{\mathcal{O}(k)}k!)$.*

**Proof.** Given an instance $(G, k)$ we prove that the algorithm outputs a $k$-cycle if and only if $G$ contains a $k$-cycle.

In one direction the algorithm answers 'Yes' and outputs a cycle. As the edges not deleted from the graph go from color-class $c_i$ to $c_{i+1(\mod k)}$, the shortest cycle in the graph is of length $k$ and since the breadth first search only test for lengths up to $k$, we will not find cycles longer than $k$. Thus if the algorithm outputs a cycle, it must be of length $k$.

For the other direction, assume in contradiction that $G$ has a $k$-cycle $S = \langle s_1, s_2, \ldots, s_k, s_1 \rangle$, while the algorithm produces a 'No' answer. Since $\mathcal{F}$ is a perfect hash family, there exists a function $f \in \mathcal{F}$ such that the vertices $\{s_1, s_2, \ldots, s_k\}$ all received different colors when $f$ was used as the coloring function. Since the algorithm tries every possible ordering of the color classes, it will try $\langle f(s_1), f(s_2), \ldots, f(s_k) \rangle$. Under this ordering, none of the edges in the cycle $S$ will be removed, and since we test every vertex of one color class $f(s_i)$, we will at some point test if there exists a cycle from $s_i$ to itself and output a 'Yes'-answer, contradicting the assumption.

**Algorithm** $k$-cycle algorithm
**Input:** A graph $G = (V, E)$ and an integer $k$.
**Output:** A subgraph of $G$, isomorphic to a cycle of size $k$, or a 'No' answer

Let $\mathcal{F}$ be a family of perfect hash functions from $\{1, \ldots, n\}$ to $\{1, \ldots, k\}$, where $|V| = n$.

**For** each function $f \in \mathcal{F}$
  Use $f$ to color $V$ using $k$ colors
  **For** each ordering $c_1, c_2, \ldots, c_k$ of the $k$ colors
    Construct a directed graph as follows:
    **for each** edge $(u, v) \in E$
      **if** $f(u) = c_i$ and $f(v) = c_{i+1(\mod k)}$ for some $i$
        **then** replace edge with arc $\langle u, v \rangle$
      **else** delete edge $uv$

    **For** all $v$ such that $f(v) = c_1$
      Use breadth first search to test if there exists a cycle $C$ from $v$ to itself of length $k$
      If such $C$ exists then output $C$ and **halt**
    **End For**
  **End For**
**End For**
Output 'No'

Figure 3.3: An algorithm for the $k$-Cycle problem.

To calculate the running time, we know by [AYZ95] that we have a perfect hash family of size $2^{\mathcal{O}(k)} \log n$. Thus, the result follows as the number of orderings of the $k$ color classes is $k!$, and the rest is a polynomial factor. □

Note that instead of investigating each possible ordering of the color classes in order to find a cycle we could use a dynamic programming strategy. This would improve the running time, but we have chosen this simpler version because we wish to emphasize the color coding part of the algorithm.

# 4

# REDUCTION RULES

In this chapter we look at various techniques that all include the application of reduction rules to transform a general instance into a manageable instance that preserves the correct solutions. This includes the second of the two most common techniques in fixed parameter algorithm design: kernelization or preprocessing. We discuss the basics first, giving some examples and showing some of the pitfalls inherent in this technique. We then continue with a more advanced version called Crown Decomposition.

## 4.1 THE BASIC REDUCTION

The fact that smaller and/or more structured instances are simpler to solve than large and/or general ones is a banality. Nevertheless, this is what we want to exploit in this technique. By gradually changing an instance, reducing it in size and imposing more structure, we seek to reach a state where it can either be decided immediately or it has been simplified sufficiently. In the context of size reduction, we say that an instance has been sufficiently reduced when the size of the instance can be bounded by a function $f$ which depends only on $k$. We call an instance that has been sufficiently reduced a *kernel*.

The main tool to facilitate the reduction is the *reduction rule*. Each reduction rule identifies a certain structure LHS (the left-hand-side) in the instance and modifies it to RHS (the right-hand-side), usually by deletion (RHS = $\emptyset$). The identified structure and resulting change is usually of a fixed size, although more complicated reduction rules exist as well.

If a reduction rule $A$ cannot be applied to an instance we say that the instance is *irreducible for* $A$. That an instance is irreducible implies that it does not have the structure the reduction rule applies to. In this way, we can use reduction rules to remove a certain structure from an instance. This allows us to shed away trivial and/or polynomial solvable parts of an instance, thus revealing the combinatorial hard core.

Normally a single reduction rule is not sufficient to reduce a problem to a kernel. We usually require a set of reduction rules, where each rule in the set removes one type of problematic structure. If a set of reduction rules is sufficient to reduce any instance of a

**Algorithm** A complete set $R$ of $i \geq 1$ reduction rules of type $R_i : LHS_i \rightarrow RHS_i$
**Input** Graph $G$, integer $k$

**while** $G$ contains some $LHS_i$
    remove $LHS_i$ from $G$ and replace by $RHS_i$
**end while**
**run** a brute force algorithm on the reduced graph

Figure 4.1: An algorithm skeleton for a complete set of reduction rules.

problem to a kernel we call it a *complete set of reduction rules*.

The process of reducing the problem in size to a kernel is called *kernelization* and can be viewed as a preprocessing step for a brute force algorithm that will operate on the reduced kernel. An algorithm skeleton for this can be seen in Figure 4.1. We will adopt the following definition of a kernelization algorithm.

**Definition 4.1.1** *Let $\mathcal{L}$ be a parameterized problem, i.e., $\mathcal{L}$ consists of pairs $(x, k)$. A* kernelization *algorithm for problem $\mathcal{L}$ is a polynomial time algorithm that reduces an arbitrary instance $(x, k)$ to an instance $(x', k')$ such that the following is satisfied: $k' \leq g(k)$ and $|x'| \leq f(k)$ where $g$ and $f$ are arbitrary functions depending only on $k$, and $(x, k) \in \mathcal{L} \iff (x', k') \in \mathcal{L}$.*

If $f$ in the above definition is a linear function, we say that we have a *linear kernel*. Likewise if $f$ is a quadratic function, we say that we a have a *quadratic kernel* and so on. The running time of the brute force algorithm is usually $\mathcal{O}^*(c^{f(k)})$ so it is always interesting to have function $f$ of as low order as possible, preferably linear. Beware, in the literature on graph problems it is common to consider a kernel to be linear if the number of vertices is linearly bounded. It could be argued that it would be more correct if a linear kernel implied that the number of vertices *and* the number of edges were both bounded by a linear function of $k$. In general, we adopt the common terminology for graph problems and consider the size of a kernel to be the number of vertices it has.

Note that all problems in the class $FPT$ have kernelization algorithms. As we will see in the proof below, this is a simple observation, but it is important to point out the relationship.

**Theorem 4.1.1** *A problem $\mathcal{A}$ is in FPT if and only if $\mathcal{A}$ has a kernelization algorithm.*

**Proof.** The fact that a problem is in FPT when it has kernelization algorithm is trivial. After we have reduced the instance to a kernel, we can solve the problem with any brute

force algorithm. Since the instance has size $f(k)$, the brute force algorithm will solve the problem in FPT time.

In the other direction, we know by definition that a problem in $FPT$ has an algorithm that decides the problem in time $\mathcal{O}(n^{\alpha}f(k))$. Now consider an instance $(x,k)$ where $|x| = n$. If $n \geq f(k)$, the algorithm runs in time $\mathcal{O}(n^{\alpha+1})$ and is polynomial, this is akin to reducing the problem to a trivial case. On the other hand, if $n \leq f(k)$ then the instance already is a problem kernel and the result holds. □

### 4.1.1 EXAMPLES OF REDUCTION RULE ALGORITHMS

**Vertex Cover**

Here we will give the classical example of a quadratic kernelization algorithm for VERTEX COVER (P7.21). It is by far the simplest algorithm using reduction rule known to us and illustrates the technique very well. We will make use of the following reduction rule, due to S. Buss [DF99].

**Reduction Rule 1** *Assume $v \in V(G)$ with $deg(v) > k$. Then $G$ has a* VERTEX COVER *of $k$ vertices if and only if $G - v$ has a vertex cover of $k-1$ vertices.*

**Proof.** One direction is trivial, if $G - v$ has a vertex cover $S$ of size $k-1$ then obviously $G$ has a vertex cover of size $k$ by adding $v$ to $S$.

In the other direction, we assume that we have a vertex cover $S$ of size $k$. Assume that $v$ is not in $S$. Since $S$ is a vertex cover, we must have $N(v) \in S$, but then $|S| \geq |N(v)| > k$, contradicting that $|S| = k$. Thus we know that $v \in S$ and then $S - v$ is a vertex cover of size $k-1$ for $G - v$. □

Armed with this reduction rule we can create the algorithm in Figure 4.2.

**Theorem 4.1.2** *The '$k^2$ kernelization algorithm for Vertex Cover' is correct and produces a kernel of $\mathcal{O}(k^2)$ vertices for vertex cover in linear time.*

**Proof.** Let us examine the reduced graph $G'$ that remains after the **for**-loop has deleted all vertices of degree more than $k$. This instance $G'$ (with the parameter $k'$) has been obtained from the input $(G,k)$ by repeatedly applying Reduction Rule 1. Thus by correctness of Reduction Rule 1 we know that $G'$ has a vertex cover of size $k'$ if and only if $G$ has a vertex cover of size $k$.

Since the reduced graph $G'$ has vertices of degree at most $k$, any vertex in $V(G')$ can cover at most $k$ edges. Thus the total number of edges a vertex cover of size $k'$ can cover

**Algorithm** A $k^2$ kernelization algorithm for Vertex Cover
**In:** A graph $G = (V, E)$, integer $k$
**Out:** A kernel of at most $k^2$ vertices or a 'No' answer

Let $k'$ = $k$
**for each** $v \in V$
  **if** deg$(v) > k$ **then**
    Delete $v$
    Decrease $k'$ by 1
  **end if**
**end for**

Delete vertices of degree 0, call the resulting graph $G' = (V', E')$

**if** $|E'| > k' \cdot k$ **then** Output 'No'
**else** Output $(G', k')$

Figure 4.2: A $k^2$ kernelization algorithm for vertex cover

is at most $k' \cdot k$, thus it is at this point safe to reject any graph $G'$ with more edges than $k' \cdot k$.

It is easy to see that the algorithm works in linear time. It simply scans through the vertices and deletes the vertices of degree more than $k$. □

### Max Cut

The next example we will give is how to obtain a kernel for MAX CUT (P7.10). For this problem we will assume that the input is not necessarily connected.

Consider a graph $G = (V, E)$ with $c$ components. Observe that the edges of a spanning forest of $G$ can be arranged such that every edge crosse one cut. It follows that $G$ has a MAX CUT of at least $|V(G)| - c$ edges, and thus for any nontrivial instance it holds that $|V(G)| - c < k$. If the graph is disconnected we can connect it with the following reduction rule:

**Reduction Rule 2** *Let $G = (V, E)$ be a disconnected graph with $c$ components, and let $v_1, v_2, \ldots, v_c$ contain one vertex from each component. We then have that $G$ has a* MAX CUT *of size at least $k$ if and only if $G'$ obtained from $G$ where $v_1, v_2, \ldots, v_c$ is replaced by $v_c$ and $N(v_c) = \bigcup_{1 \leq i \leq c} N(v_i)$ has a* MAX CUT *of size at least $k$.*

This is true as there for any optimal cut it is possible to arrange the vertices in each component such that $v_1, v_2, \ldots, v_c$ is on the same side of the cut (if needed a components vertices can be flipped along the cut).

After this reduction rule has been applied we have that $|V(G')| = |V(G)| - c + 1$. Inserting this equation into $|V(G)| - c < k$ we get that $|V(G')| \leq k + 1$ and we have the following theorem:

**Theorem 4.1.3** MAX CUT *has a kernel of size* $k + 1$ *vertices.*

Note that this kernel will lead to an algorithm that is far from optimal. [MR99, P04, LS05] all obtain a bound of $2k$ edges for MAX CUT, and by using the strong exact algorithm [FK02] for MAX CUT with running time $\mathcal{O}^*(2^{|E|/4})$, we obtain a FPT running time for Max Cut of $\mathcal{O}^*(2^{k/2})$.

#### Planar Independent Set

For some problems obtaining a kernel is trivial. In the following example we bound the size of the kernel without applying any reduction rules.

**Theorem 4.1.1** DOMINATING SET ON CUBIC GRAPHS (P7.4) *has a kernel of* $4k$ *vertices.*

**Proof.** Observe that since the maximum degree of a vertex is 3, no vertex can dominate more than 4 vertices. Thus $k$ vertices can dominate no more than $4k$ vertices. This gives an immediate upper bound on any 'Yes' instance. □

The proof above leads to a linear kernel of $4k$ vertices and $12k$ edges. For many problems this would be a very good result, but here it is terrible. By the same argument as in the proof we see that no cubic graph has a dominating set of size less than $n/4$. Thus for any non-trivial problem instance we have $k \geq n/4$ and thus $4k \geq n$, and the bound $4k$ obtained from the kernel is larger than the size of the instance itself.

This shows that it is important to be aware of the lower and upper bounds on interesting instances of the problem one is working on. This can be of great help in finding trivial kernels and estimating the quality of a suggested kernel.

## 4.2 CROWN DECOMPOSITION

The reduction rules we have seen so far have all been of fixed size and have focused on a local structure in an instance, for example a vertex with particularly high or low degree.

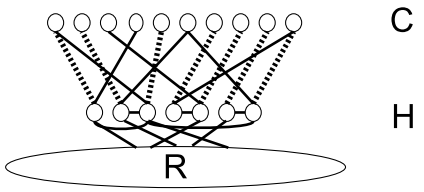


Figure 4.3: Example of a crown decomposition. The matched edges are dashed

In the last few years, there has been some focus on reduction rules that do not follow this pattern. In this section we will carefully study one type of these reduction rules, those that are based on *crown decompositions*.

**Definition 4.2.1** *A* crown decomposition *of a graph $G = (V, E)$ is a partitioning of $V$ into sets $C, H,$ and $R$, where $C$ and $H$ are nonempty, such that the following properties are satisfied:*

1. *$C$ is an independent set.*
2. *No edge between a vertex in $C$ and a vertex in $R$.*
3. *There exists an injective map $m : H \to C$, such that $m(a) = b$ implies that $(a, b) \in E$ is an edge.*

An illustration of a crown decomposition of a graph can be seen in Figure 4.3.

The main reason that crown decompositions are so useful, is that we have a very nice lemma that tells us when we can find a crown decomposition in a graph. This lemma is due to Chor, Fellows, and Juedes and can be found in [CFJ04].

**Lemma 4.2.1** *If a graph $G = (V, E)$ has an independent set $I \subseteq V(G)$ such that $|N(I)| < |I|$, then a crown decomposition $(C, H, R)$ for $G$ such that $C \subseteq I$ can be found in time $\mathcal{O}(|V| + |E|)$.*

In reduction algorithms based on crown decompositions, we often see reduction rules that do not necessarily reduce the graph in size but rather modifies the graph so that it allows a larger independent set $I$. Eventually the lemma above is invoked, and we can use the independent set $I$ to obtain a crown decomposition. This crown decomposition is then used to obtain a size reduction of the graph. A good example of this can be seen in Chapter 8.

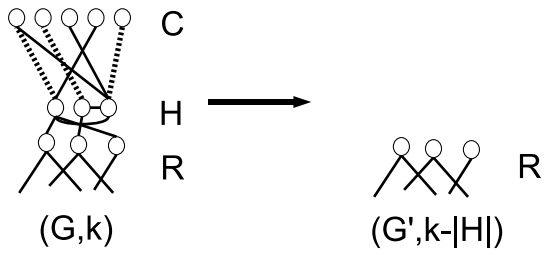


Figure 4.4: A crown reduction rule for VERTEX COVER, reducing the size of both the graph and the parameter. Here $LHS = (C \cup H)$ and $RHS = \emptyset$.

Although crown reduction rules were independently discovered by Chor, Fellows, and Juedes [CFJ04] one should note that a similar type of structure has been studied in the field of boolean satisfiability(SAT) under the name of 'autarchies' [K00, K03]. As we know the main goal of a satisfiability problem is to find a truth assignment for a clause set over a set of variables. An *autarky* is a partial truth assignment (assigns true/false to only a subset of the variables) such that each clause that contains a variable from the variables determined by the partial truth assignment is satisfied.

In a matching autarky we require in addition that the clauses satisfied and the satisfying variables form a matching cover in the natural bipartite graph description of a satisfiability problem. It is easy to see that the matching autarky is a crown decomposition in the bipartite graph.

The main protagonist for autarchies is Oliver Kullmann, who has developed an extensive theory on several different types of autarchies. Unfortunately, aside from the *matching autarchies* described above, the other types of autarchies do not transfer easily to general graphs.

### 4.2.1 EXAMPLE OF A CROWN REDUCTION RULE

The simplest example of a reduction rule using a crown decomposition is perhaps the following folklore algorithm which applies to the problem VERTEX COVER (P7.21), see Figure 4.4.

**Lemma 4.2.2** *Given a crown decomposition $(C, H, R)$ of a graph $G = (V, E)$, then $G$ has a* VERTEX COVER *of size $k$ if and only if $G' = G[V - (C \cup H)]$ has a* VERTEX COVER *of size $k' = k - |H|$.*

**Proof.** Assume in one direction that $G'$ has a VERTEX COVER $S'$ where $|S'| = k - |H|$. We can then create a VERTEX COVER $S$ for $G$ by adding $H$ to the vertex cover, that is

**Algorithm** Crown Kernelization
**Input:** A graph $G = (V, E)$ and an integer $k$.
**Output:** A kernel of size at most $4k$ or a 'No' answer

**do** Create a maximal matching $M$ in $G$
  **if** $|V(M)| > 2k$ **then output** answer 'No' and **halt**
  **else**
    **if** $|V(G) - V(M)| \leq 2k$ **then output** $(G, k)$ and **halt**
    **else**
      Create crown decomposition $(C, H, R)$
      Let $G = G[V - (C \cup H)]$, and $k = k - |H|$
**repeat**

Figure 4.5: A $4k$ kernelization algorithm for vertex cover.

$S = S' \cup H$. Observe that $H$ is incident to all edges in $E(G) - E(G')$.

In the other direction we assume that $G$ has a VERTEX COVER $S$ of size $k$. Due to the matching between $H$ and $C$ we have $|S \cap (H \cup C)| \geq |H|$. (A vertex cover has to pick one vertex incident to each matched edge.) Thus the number of vertices in $S$ covering edges not incident to $H \cup C$ is at most $k - |H|$ and the result follows. □

It is this reduction rule that is used to produce the $2k$ vertex kernel mentioned in Chapter 5. It is also possible to reach a slightly weaker kernel of $4k$ with a simple Win/Win argument. We will give this algorithm here to show the reduction rule in practice.

**Lemma 4.2.3** *The algorithm Crown Kernelization either terminates with a correct no answer or produces a kernel of at most $4k$ vertices for* VERTEX COVER.

**Proof.** To see that the algorithm always terminates, observe that the graph either gives an output or reduces the graph. Since we can have at most $\mathcal{O}(n)$ reductions, the algorithm will eventually terminate.

We first show that a 'No' answer is always correct. The algorithm will only output 'No' if there is a maximal matching $M$ in $G$ where $|V(M)| > 2k$. Since we have to pick at least one vertex from each edge, the vertex cover for this graph is greater than $k$.

The algorithm modifies the instance $(G, k)$, so we have to make sure we do not introduce or remove any solutions. At any stage of the algorithm the current graph and parameter has been obtained by repeated applications of the crown reduction rule. By Lemma 4.2.2,

we are thus guaranteed that the reduced instance is a 'Yes'-instance if and only if the input instance is a 'Yes'-instance.

We have $|V(G)| = |V(M)| + |V(G) - V(M)| \leq 2k + 2k = 4k$, so the algorithm outputs only graphs of at most $4k$ vertices. Thus, the lemma is correct. □

### 4.2.2 THE COMPLEXITY OF CROWN DECOMPOSITION

It has been shown that one can determine in polynomial time if a graph has a crown decomposition [ALS05]. The algorithm given in that paper will also find a crown decomposition in polynomial time if one exists. However, the following theorem shows that it is harder to find crowns if we give an additional requirement on the size of $H \cup C$. This result has to our knowledge not appeared in the literature.

SIZED CROWN (P7.18)
Instance: $G = (V, E)$
Parameter: Positive integer $k$
Question: Does $G$ have a crown decomposition where $|H \cup C| = k$?

**Theorem 4.2.1** SIZED CROWN *is NP-complete.*

Before we begin the proof we describe two gadgets.

$A_n$: A complete bipartite $K_{n,n}$
$A_n^{+1}$: A complete bipartite graph $K_{n,n}$ and one vertex $x$ universal to one partition, for a gadget $G' = A_n^{+1}$, if $V' = V(G')$ we write $x(V')$ to indicate as this special 'entrance' vertex $x$.

One should note that both gadgets can form a crown on its own (one partition in $H$, and one in $C$) and also note that if a vertex in a gadget is in $C$ in a crown decomposition $(C, H, R)$, then this implies that all vertices universal to this gadget are in $H$, due to crown-separation property.

**Proof.** It is obvious that SIZED CROWN is in NP, to prove hardness we will reduce the problem from HITTING SET, given a collection of sets $Q$ over a set of elements $X$, is there a subset $S \subseteq X$, $|S| = k$ such that $\forall Q_i \in Q, S \cap Q_i \neq \emptyset$.

Construct $G$ with the following set of vertices:
$V_X = \{v_{x_i} \mid x_i \in X\}$
$V'_X = \{v'_{x_i} \mid x_i \in X\}$
$V_Q = \{v_{Q_i} \mid Q_i \in Q\}$

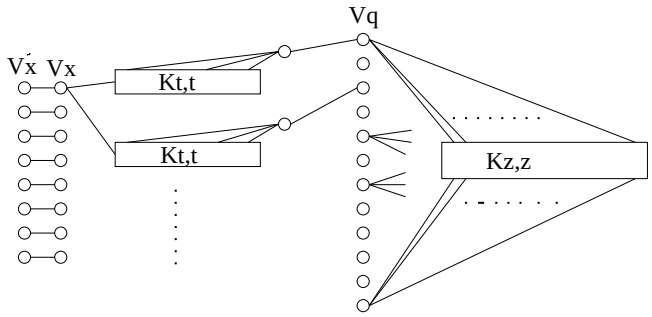


Figure 4.6: An incomplete illustration of G, here element 1 appears in set 1 and 3

and the following gadgets (with internal edges):
$V_A^{X_i,Q_j} = A_t^{+1}, \forall x_i, Q_j$ where $x_i \in Q_j, t = |X| + |Q|$
$V_B = A_z$, and $z = |V_X \cup V'_X \cup V_Q| + \sum_{x_i \in Q_j} V_A^{X_i,Q_j}$ (the rest of the graph)

and edges:

$E_1 = \{v_{x_i} v'_{x_i} \mid v_{x_i} \in V_X, v'_{x_i} \in V'_X\}$
$E_2 = \{v_{x_i} u \mid v_{x_i} \in V_X, u \in V_A^{X_i,Q_j} \setminus x(V_A^{X_i,Q_j})\}$
$E_3 = \{x_i v_{Q_j} \mid v_{Q_i} \in V_Q, x_j = x(V_A^{X_i,Q_j})\}$
$E_4 = \{v_{Q_i} u \mid v_{Q_i} \in V_Q, u \in V_B\}$

see also Figure 4.6 for clarity.

We now prove that $(Q, X)$ has a HITTING SET of size $k$ if and only if $G$ has a crown decomposition where $k' = |H \cup C| = 2k + 2|Q| + 2z + 2t|Q|$.

($\Rightarrow$:) Assume that $G$ has a HITTING SET $S'$ of size $k$. We will construct a crown decomposition in the following manner. For each element $x'_i \in S'$, let the corresponding vertex $v_{x_i} \in H$, and let $v'_{x_i} \in C$. (in total $2k$ vertices)

For all $v_{Q_j} \in V_Q$, let $v_{Q_j}$ be in $H$. If the corresponding set $Q_j$ was covered by some element $x_i$ (if more than one element covers $Q_j$, then select one arbitrarily), let $x(V_A^{X_i,Q_j})$ be in $C$ and let $V_A^{X_i,Q_j}$ be in $(H \cup C)$ as needed( in total $2|Q| + 2t|Q|$).

Finally let $V_B$ be in $(H \cup C)$ as needed (in total $2z$ vertices). It is easy to verify that (1) there are no edges from a vertex in $C$ to a vertex in $V - H$ (2) Each vertex in $H$ has a private neighbor. This is a crown and it has $k' = 2k + 2|Q| + 2z + 2t|Q|$ vertices.

($\Leftarrow$:) Assume that $G$ has a crown of size exactly $2k + 2|Q| + 2z + 2t|Q|$.

Observe that $V_B$ is $2/3$ of the vertices of $G$ thus $V_B \cap C \neq \emptyset$. This implies that $V_B \subseteq (C \cup H)$ and that $V_Q \subseteq H$. Since the vertices in $V_Q$ must be matched, at least $q \geq |Q|$ different $A_n^{+1}$ gadgets will have their 'entrance vertex' in $C$, thus at least $l$ complete bipartite graphs are in $(C \cup H)$.

If $l > |Q|$ then $|C \cup H| \geq 2z + |Q| + 2tl \geq 2z + |Q| + 2t(|Q|+1) \geq 2z + |Q| + 2t|Q| + 2t \geq 2z + |Q| + 2t|Q| + 2(|X| + |Q|) \geq 2z + 2|Q| + 2t|Q| + 2|X| + |Q| > k'$ contradicting. Thus $|Q| \leq l \leq |Q|$ holds.

The neighboring vertices in $V_X$ of each of these gadgets will thus be in $H$, in total $r$, and each of them must match somewhere (to $V'_X$ or a to a gadget). Matching to a gadget causes the crown to be too large thus each vertex must match to its neighbor in $V'_x$. If $r > k$ then the size of the crown is more than $k'$, thus the elements corresponding to the $r \leq k$ vertices in $V_X$ is a HITTING SET of size at most s $k$(if $r < k$, one can add arbitrary elements to reach a hitting set of size exactly $k$).

□

Note that this proof does not show that SIZED CROWN is W[1]-hard. HITTING SET is W[1]-hard, but the reduction above is not an FPT-reduction as the parameter in the crown decomposition we reduce to is not solely dependent on the parameter in the HITTING SET Problem we reduce from. To our knowledge, the question if SIZED CROWN is FPT or W[1]-hard is still open, and deciding it would require a different proof idea than we have used here.

# 5

# FPT BY INDUCTION

Here we present a technique closely related to mathematical induction. We show that if we are provided a solution for a smaller instance $(G, k)$ we can for some problems use the information to determine the solution for one of the larger instances $(G + v, k)$ or $(G, k + 1)$.

We will argue that the two techniques Iterative Compression and Extremal Method are actually two facets of this inductive technique, depending on wether the problem is a minimization problem or a maximization problem.

## 5.1 THE BASICS

In his book *Introduction to Algorithms* [M89] Udi Manber shows how induction can be used as a design technique to create remarkably simple algorithms for a range of problems. He suggests that one should always consider how to solve a small instance and then how to construct a solution based on solutions to smaller problems.

For example this technique leads to the well known INSERTION SORT algorithm when used on the SORTING PROBLEM (P7.19), as follows:

It is trivially true that we can arrange sequences of length 1 in increasing order. Assuming that we could sort sequences of length $n - 1$, we can sort sequences of $n$ elements by first sorting the first $n - 1$ elements and then inserting the last element at its correct place. Since the insertion takes $\mathcal{O}(n)$ time and it is done once for each element we have a total running time of $\mathcal{O}(n^2)$.

The core idea of the technique is based on using the information provided by a solution for a smaller instance. When an instance contains both a main input and a parameter input, we must be clear about what we mean by 'smaller' instances. Let $(G, k)$ be an instance, where $G$ is the main input and $k$ the parameter. We can now construct three distinctly different 'smaller' instances $(G-v, k)$, $(G, k-1)$ and $(G-v, k-1)$. Which one of these to use ? Clearly, trying to solve $(G, k - 1)$ for a minimization problem or $(G - v, k)$ for a maximization problem is harder than the original problem and can be disregarded. The

combination of the two $(G - v, k - 1)$ may have some merit in either case.

In the next section we show that using smaller instances of the type $(G - v, k)$ is very suitable for minimization problems and leads to a technique known as Iterative Compression. Then we show that using smaller instances of the type $(G, k - 1)$ can be used to construct algorithms for maximization problems and is in fact the technique known as the Extremal Method.

## 5.2 FOR MINIMIZATION - ITERATIVE COMPRESSION

In this section we present Iterative Compression. This technique works well on minimization problems. We will try to improve smaller instances of the type $(G - v, k)$, i.e., the graph instance minus one vertex.

Since it is an inductive method we can assume that we can compute the solution for the smaller instance. Since our problems are decision problems, the solution can take two forms, either a 'Yes'-answer or a 'No'-answer. We now wish to use the information provided by the answer to compute the solution for $(G, k)$. Because of the nature of this technique we can assume that the algorithms are constructive and thus for a 'Yes'-instance we also have a certificate that verifies that the instance is a 'Yes'-instance (recall that NP is the class of problems with such certificates). However, for a 'No'-answer we receive no extra information, but we still need to be able to calculate the solution for $(G, k)$.

A class of problems where 'No'-answers carry sufficient information is the class of *monotone* problems. We will say that a problem is monotone if the 'No'-instances are closed under element addition. Thus if a problem is monotone we can immediately answer 'No' for $(G, k)$ if $(G - v, k)$ is a 'No'-instance.

Given a monotone problem we can use an algorithm skeleton as seen in Figure 5.1. The algorithm will recursively call itself at most $|V(G)|$ times, thus the running time of an algorithm of this type is $\mathcal{O}(n)$ times the time it takes to compute a solution given a smaller solution.

Note that some minimization problems are not monotone. Consider for instance DOMINATING SET (P7.3), where the addition of a universal vertex always changes a 'No' answer to 'Yes' (unless $k \geq n$). For such problems we believe that this technique is ill suited.

### 5.2.1 EXAMPLE OF 'ITERATIVE COMPRESSION'

We will give an algorithm for the problem FEEDBACK VERTEX SET using the technique described in the previous section. FEEDBACK VERTEX SET (P7.5) is a monotone problem as adding new vertices will never help to cover existing cycles. Consider 'Feedback

**Algorithm skeleton** Iterative Compression
**Input:** Graph $G$, integer $k$

**if** $|V(G)| < k$ **then run** brute force algorithm, return answer and **halt**
Choose arbitrary vertex $v$
Recursively call problem on instance $(G - v, k)$, obtain solution $S$
**if** $S$ = 'No' **then** answer 'No'
**else** Compute in FPT time a solution for $(G, k)$ based on $S$

Figure 5.1: An algorithm skeleton for Iterative Compression, this technique assumes that the problem is monotone.

Vertex Set Algorithm' in Figure 5.2.

**Algorithm** 'FVS'
**Input:** Graph $G$, integer $k$

**if** $|V(G)| \leq k$ **then return** 'Yes' and **halt**
Choose arbitrary vertex $v \in G$
$S = FVS(G - v, k)$
**if** $S$ = 'No' **then** answer 'No' and **halt**

Let $T = G - (S \cup v)$, note that $T$ is a forest.
Create tree decomposition $(X, I)$ of $T$.
Add $S \cup v$ to each bag in $X$ to obtain tree decomposition $(X', I)$ of width at most $k + 2$.
**run** Treewidth algorithm for feedback vertex set on tree decomposition $(X', I)$
**output** answer from treewidth algorithm.

Figure 5.2: An inductive algorithm for FEEDBACK VERTEX SET.

**Theorem 5.2.1** *Algorithm 'FVS' is correct, and solves* FEEDBACK VERTEX SET *in FPT time.*

**Proof.** We will first prove that the algorithm does not incorrectly decide the answer before running the treewidth sub routine. If the algorithm answers 'Yes' because $|V(G)| \leq k$, it is correct as we can select $V$ as our feedback vertex set. If the algorithm answers 'No' because $(G - v, k)$ is a 'No'-instance, it is correct as FEEDBACK VERTEX SET is a monotone problem.

We assume the treewidth subroutine is correct so it remains to show that the algorithm computes a tree decomposition of the graph with bounded treewidth. The algorithm computes the graph $T = G - (S \cup v)$ which is a forest, and it is easy to construct a tree decomposition of width 1 from this forest, having one bag for each edge. The algorithm then adds $S \cup v$ to each bag to obtain a correct tree decomposition of width $k + 2$.

It follows from results in [ALS91] that FEEDBACK VERTEX SET is solvable in FPT time if the treewidth is the parameter. This gives the desired result. □

### 5.2.2 HOW IS IT USED IN PRACTICE?

Two papers that use this type of induction are [RSV03] and [DFRS04]. In [RSV03], Reed, Smith, and Vetta managed to show that the problem $k$-ODD CYCLE COVER (P7.13) (is it possible to delete $k$ vertices from $G$ to obtain a bipartite graph) is in FPT, thus settling a long open problem in the field. By the induction hypothesis, assume that we can determine if $(G - v, k)$ is a 'No' or a 'Yes' instance for $k$-ODD CYCLE COVER. The induction step is then in two parts. First, the trivial part: to show that if $(G - v, k)$ is a 'No'-instance then the inclusion of another vertex cannot improve the situation. The second part deals with the situation when we have a positive certificate $S$ consisting of the $k$ vertices to be deleted from $(G - v, k)$ to make it bipartite. If we have such a certificate $S$, we can conclude that $S \cup \{v\}$ is a solution of size at most $k + 1$ for $G$. The authors then show, by a fairly complicated argument which we will not explain here, that given a solution of size at most $k + 1$ it is possible to determine in FPT time whether or not there exists a solution of size $k$.

A very similar use of induction is seen in [DFRS04] where Dehne, Fellows, Rosamond and Shaw give a $2k$ kernel for VERTEX COVER (P7.21) without using the complicated Nemhauser-Trotter results [NT75]. By induction assume we can determine if $(G - v)$ has a $k$-VERTEX COVER, if no such cover exists we cannot find one for $G$ either. On the other hand if $(G - v)$ has a $k$-VERTEX COVER $S$ then $S \cup \{v\}$ is a $k + 1$-VERTEX COVER for $G$ and by implication has a $n - (k + 1)$-INDEPENDENT SET. As long as $|V(G)| > 2k + 2$ we know by Lemma 4.2.1 that $G$ has a crown decomposition, which in turn leads to a reduction in $G$ as seen in Lemma 4.2.2. This reduction continues until the graph has size at most $2k + 2$, at which point we can test for $k$-VERTEX COVER using a brute force algorithm.

## 5.3 FOR MAXIMIZATION - THE EXTREMAL METHOD

For maximization problems we consider smaller instances of the type $(G, k - 1)$. Thus we will induct on $k$ instead of on $n$. Most maximization problem are trivial for $(G, 0)$.

For the same reasons as before we need monotonicity in the problem, but since we are inducting on the parameter we will instead require that the 'No'-instances are closed under

parameter increment. That is, if instance $(G, k)$ is a 'No'-instance then $(G, k')$ is also a 'No'-instance for all $k' > k$. If a problem satisfies this we say it is *parameter monotone*.

For a parameter monotone problem we can modify the algorithm skeleton from the previous section to obtain an inductive algorithm for maximization problems, see Figure 5.3.

---

**Algorithm skeleton** for parameter monotone problems
**Input:** Graph $G$, integer $k$

**if** $k = 0$ **then run** brute force algorithm, return answer and **halt**
Inductively call problem on instance $(G, k-1)$, obtain solution $S$
**if** $S$ = 'No' **then** answer 'No'
**else** Compute in FPT time a solution $S'$ for $(G, k)$ based on $S$ and return $S'$

---

Figure 5.3: An algorithm skeleton for parameter monotone problems.

One way to compute a solution for $(G, k)$ based on the certificate $S$ for the 'Yes'-solution for $(G, k-1)$ is to make rules of the following type:

> *Either solution $S$ can be modified to be a solution for $(G, k)$ or $G$ has property $P$.*

Thus we either prove that $G$ is a 'Yes'-instance for $(G, k)$ and can terminate, or $G$ has a specific property $P$. By creating a set of rules $R$ of this type we hope to impose enough structure to prove a lemma of this type:

> *If no rule in $R$ applies to $(G, k)$ then $|V(G)| < f(k)$.*

The set $R$ of rules can be augmented with normal reduction rules to remove unwanted structures.

What we have described above is equivalent to the *Algorithmic Version* of the *Method of Extremal Structure* as described in Elena Prieto's PhD thesis [P05].

The regular *Method of Extremal Structure* tries to gradually improve a set of reduction rules $R$ until it is possible to prove a *boundary* lemma of the following type:

> *If no rule in $R$ applies to $(G, k)$ and $(G, k)$ is a 'Yes'-instance and $(G, k+1)$ is a 'No'-instance, then $|V(G)| \leq f(k)$*

Given such a boundary lemma and the fact that the problem is a *parameter monotone* maximization problem we can immediately prove a *kernelization* lemma saying

> If no rule in $R$ applies to $(G, k)$ and $|V(G)| > f(k)$, then $(G, k)$ is a 'Yes'-instance.

When trying to prove the boundary lemma we consider the class

> $E_R(k) = \{G \mid (G, k)$ is irreducible under $R, (G, k)$ is a 'Yes'-instance, and $(G, k + 1)$ is a 'No'-instance $\}$.

If $\max\{|V(G)| \mid G \in E_R(k)\}$ is a function of $k$ we can prove the boundary lemma, otherwise we need to strengthen our reduction rules.

We can find new reduction rules by considering one of the graphs with the largest number of vertices in $E_R(k)$. We can in this way often identify crucial structures that need to be taken care of with reduction rules before we can proceed.

### 5.3.1 HOW IS IT USED IN PRACTICE?

The Extremal method has been successfully applied to a range of problems and we will give a list with references here: MAX CUT [P04], MAX LEAF SPANNING TREE [P05], NON-BLOCKER [P05], EDGE-DISJOINT TRIANGLE-PACKING [MPS04], $K_{1,s}$-PACKING [PS04], $K_3$-PACKING [FHRST04], SET SPLITTING [DFR03], and MAX INTERNAL SPANNING TREE [PS05].

For an example of the extremal method we refer the reader to Chapter 9 where we use the Extremal Method to obtain a quadratic kernel for $K_{1,s}$-packing.

# 6

# WIN/WIN

Imagine that we solve our problem by first calling an FPT algorithm for another problem and use both its 'Yes' and 'No' answer to decide in FPT time the answer to our problem. Since we then 'win' if its answer is 'Yes' and we also 'win' if its answer is 'No', this is called a 'Win/Win' situation.

In this chapter we focus on this technique, which is in our opinion one of the strongest techniques for creating parameterized algorithms. Indeed, according to [DH05] the only known algorithms with sub-exponential running time $\mathcal{O}^*(c^{\sqrt{k}})$ are algorithms based on treewidth and branchwidth, and these fall into the Win/Win category.

## 6.1 BASIC WIN WIN - BEST OF TWO WORLDS

In Win/Win we make use of a relationship between two parameterized problems $A$ and $B$. We can for some problems show that an instance either is a 'Yes'-instance for problem $A$ or a 'Yes'-instance for problem $B$. This can be useful for constructing an algorithm for $A$ if we can decide problem $B$ in FPT time.

Consider the algorithm skeleton shown in Figure 6.1. We run our decision algorithm $\Phi$ for problem $B$ on input $I$. If $\Phi$ returns a 'No' answer, we know by the relationship between $A$ and $B$ that $A$ is a 'Yes'-instance(this can sometimes yield a non-constructive result where we know that $A$ is a yes-instance but we have no certificate). Otherwise we have obtained the knowledge that $I$ is a 'Yes'-instance for $B$, and if $\Phi$ is constructive, we also have a certificate. We then proceed to solve $A$ with the extra information provided. In the following subsection, we give examples of how this has been done in the literature.

### 6.1.1 EXAMPLES OF WIN/WIN

To our knowledge this technique was not given a self-standing discussion until [PS03] and then under the name 'Either/Or' (in relation only to vertex cover) and in [F03] where it was given the name Win/Win. Thus, earlier papers using this technique will not point out that a Win/Win technique is used.

**Algorithm:** A Win/Win skeleton for problem $A$ using problem $B$
**Input:** Graph $G$, parameter $k$
**run** FPT algorithm $\Phi$ for $B$ on $(G, k)$
**if** $(G, k)$ is a 'No'-instance for $B$ **then** Output 'Yes'
**else run** use structure given by $\Phi$ to solve $A$ on $G$.

Figure 6.1: The classical Win/Win algorithm structure ,although other Yes/No relationships between $A$ and $B$ may be used.

In the literature on parameterized algorithms there are several notable occurrences of a Win/Win strategy. Many problems can be related to TREEWIDTH (P7.20) or BRANCHWIDTH (P7.1) and this is of great interest as there are many powerful algorithms that solve NP-hard problems in FPT time if the parameter is the treewidth or the branchwidth of the input graph.

One example of this is PLANAR DOMINATING SET (P7.16). We give an FPT algorithm for PLANAR DOMINATING SET by showing that a planar graph with a $k$-dominating set has low branchwidth. One way to prove this is via outerplanarity, as done in [AFN01].

**Definition 6.1.1** *A graph $G$ is* 1-outerplanar *if there is a crossing-free embedding of $G$ in the plane such that all vertices are on the exterior face. A graph $G$ is $r$-*outerplanar *if it has a planar embedding such that if all vertices on the exterior face are deleted, each connected components of the remaining graph is $r'$-outerplanar for some $r' < r$.*

Vertices in $r$-outerplanar graphs are thus arranged into $r$ layers and since a vertex can only dominate vertices on its own and its two neighboring layers we have

**Observation 6.1.1** *A planar graph that has a $k$-dominating set is $r$-outerplanar for some $r \leq 3k$.*

Due to Hjortås [H05] we have

**Theorem 6.1.1** *Let $G$ be an $r$-outerplanar graph. Then $G$ has branchwidth at most $2r+1$.*

Note that a weaker bound of $3r + 1$ follows from an old result by Bodlaender [B86]. Together with Observation 6.1.1 this gives

**Corollary 6.1.1** *A planar graph that has a $k$-dominating set has branchwidth at most 6k+1.*

Computing the branchwidth of a planar graph is polynomial, using the 'Ratcatcher' algorithm [RS94]. Using this we can construct a Win/Win algorithm for PLANAR DOMINATING SET as seen in Figure 6.2. The running time of this algorithm is dependent on the dynamic programming algorithm for DOMINATING SET on graphs with bounded branchwidth.

---

**Algorithm** Win/Win for PLANAR DOMINATING SET
**Input:** A graph $G = (V, E)$ and an integer $k$.
**Output:** A 'Yes' or a 'No' answer

Run the Ratcatcher algorithm to test if branchwidth is more than 6k+1.
**if** branchwidth of $G > 6k + 1$ **then** Output 'No' and **halt**
**else run** MINIMUM DOMINATING SET algorithm for graphs of bounded branchwidth

---

Figure 6.2: A Win/Win algorithm for Planar Independent Set

**Theorem 6.1.2** *[FT03]* MINIMUM DOMINATING SET *on graphs with branchwidth at most* $bw$ *can be solved in time* $\mathcal{O}^*(2^{3\log_4 3bw})$

**Corollary 6.1.2** *Algorithm 'Win/Win for* PLANAR DOMINATING SET*' is correct and has running time* $\mathcal{O}^*(2^{14.265\,k})$.

**Proof.** Observe that the algorithm is correct when it answers 'No' when the branchwidth $bw$ is greater than $6k + 1$, as this follows directly from Corollary 6.1.1.

Otherwise the algorithm calls the MINIMUM DOMINATING SET algorithm from [FT03] and answers accordingly.

The running time is $\mathcal{O}(2^{14.265\,k})$ as the ratcatcher subroutine is polynomial and we run the MINIMUM DOMINATING SET algorithm with branchwidth at most $6k + 1$. □

It is possible to show a much stronger relationship between the size of a PLANAR DOMINATING SET and treewidth. In [AFN01] it is shown that a planar graph that has a $k$-dominating set has treewidth at most $6\sqrt{34}\sqrt{k}$, giving an algorithm of running time $\mathcal{O}(c^{\sqrt{k}})$. As mentioned before it is only Win/Win algorithms of this type that are known to give parameterized algorithms with subexponential running time for NP-hard problems. The result in [AFN01] requires some work, and we will instead give a more elegant proof of a $c^{\sqrt{k}}$-algorithm from [FT04], which is based on the following result from [RS91].

**Theorem 6.1.3** *Let* $t \geq 1$ *be an integer. Every planar graph with no* $(t \times t)$*-grid minor has branchwidth* $bw(G) \leq 4t - 3$.

We can then prove the following Lemma.

**Lemma 6.1.1** *If $G$ has branchwidth $bw(G) > 8\sqrt{k}+5$ then $G$ has no* PLANAR DOMINATING SET *of size $k$.*

**Proof.** Let $t = 2\sqrt{k}+2$, since $8\sqrt{k}+5 = 4t-3$ we know by Theorem 6.1.3 that $G$ has a $(2\sqrt{k}+2)\times(2\sqrt{k}+2)$ grid $M$ as a minor.

Consider an embedding of graph $G$ in the plane. We will now try to calculate a minimum bound on the dominating set needed to dominate the vertices in $M$. A vertex $v$ could potentially dominate the entire outer face of the grid, but any vertex in any face of the grid, or any vertex of the grid itself cannot dominate more than $4$ of the vertices in the $(2\sqrt{k})\times(2\sqrt{k})$ *inner* grid, that is $M$ minus it's outer face. Thus we need $2\sqrt{k}\cdot 2\sqrt{k}/4 = k$ vertices to dominate the inner grid, plus at least one to dominate the outer face of $M$. This proves the lemma. □

Thus we could modify the input Ratcatcher subroutine in Figure 6.2 to check if the graph has branchwidth $8\sqrt{k}+5$. This would give the following

**Theorem 6.1.4** PLANAR DOMINATING SET *can be solved in time $\mathcal{O}^*(2^{19.02\sqrt{k}})$.*

[FT04] proves the even stronger result, that a planar graph $G$ with $bw(G) > 3\sqrt{4.5}\sqrt{k}$ implies no PLANAR DOMINATING SET of size $k$, giving a $\mathcal{O}^*(2^{15.13\sqrt{k}})$ algorithm for the problem, which is the fastest known.

## 6.2 GRAPH MINOR THEOREM

The famous Graph Minor Theorem can be used to prove membership for FPT. Note that the proofs are nonconstructive and the algorithms contain hidden huge constants.

**Definition 6.2.1** *A graph $H$ is a* minor *of a graph $G$, denoted $H \preceq_m G$, if a graph isomorphic to $H$ can be obtained from $G$ using the following three operations repeatedly.*

1. *Deleting an isolated vertex*
2. *Deleting an edge*
3. *Edge contraction*

The relation $\preceq_m$ forms an ordering of finite graphs, this ordering is clearly a quasi-order, i.e, $\preceq_m$ is reflexive and transitive. However, the key result in this technique is the following famous graph minors theorem due to Robertson and Seymour [RS99].

**Theorem 6.2.1** *Finite graphs are well-quasi-ordered by the minor ordering.*

A quasi-order is a *well-quasi ordering* if there is no infinite antichain. An *antichain* is an infinite set of elements, no two of which are comparable in the ordering.

In order to see how this can be used to prove the existence of FPT algorithms, consider a problem $A$ with the property that for any $k$ the 'Yes'-instances are closed under minors. In other words if a graph $G$ is a 'Yes'-instance then $H$ such that $H \preceq_m G$ is also a 'Yes'-instance. The pair $(A, k)$ forms the minor-closed graph class $A_k$ consisting of all graphs $G$ such that $(G, k)$ is a 'Yes'-instance. Now consider the graph class Minimal Forbidden Minors of $A_k$, denoted $MFM(A_k)$.

$$MFM(A_K) = \{G \mid G \notin A_k \text{ and } (\forall H \preceq_m G, H \in A_K \vee H = G)\}$$

By definition, this class is an anti-chain of the $\preceq_m$ ordering of finite graphs.

Theorem 6.2.1 tells us that $MFM(A_k)$ is a finite set and its size is clearly only dependent on $k$. Note that we can test if an input graph $G$ is a member of $A_k$ by checking if it contains as a minor any of the graphs in $MFM(A_k)$. The main subroutine is this problem:

MINOR ORDER TEST (P7.12)
Instance: Graphs $G$ and $H$
Parameter: $|V(G)|$
Question: Is $H \preceq_m G$?

MINOR ORDER TEST is proven FPT by Robertson and Seymour [RS99]. Putting it all together we get the algorithm skeleton in Figure 6.3.

This can be considered a Win/Win algorithm as we relate the problem we wish to solve to the problem of checking if one of the forbidden minors appear in our problem instance.

Note that since the forbidden minors change as $k$ grows, the algorithm itself also changes (it needs to know the forbidden minors for each $k$). Thus we need one algorithm for each fixed $k$.

**Algorithm** An algorithm skeleton for the Graph Minor technique
**Input:** A graph $G = (V, E)$ and an integer $k$.
**Output:** A 'Yes' or a 'No' answer

Let $MFM$ be the set of forbidden minors for the specified problem and the given $k$.
**for** each minor $H \in MFM$
  **if** $H \preceq_m G$ **then answer** 'No' and **halt**
**end for**

**answer** 'Yes'

Figure 6.3: An algorithm skeleton for using graph minors which is applicable when the class $\{G \mid (G, k)$ is a 'Yes'-instance$\}$ is closed under minors.

### 6.2.1 EXAMPLE OF A GRAPH MINOR ALGORITHM

Applying this technique is very simple. It suffices to check whether the 'Yes'-instances are preserved under operation $1, 2,$ and $3$ in Definition 3.

Consider the problem FEEDBACK VERTEX SET (P7.5).

**Observation 6.2.1** *If $(G, k)$ is a 'Yes'-instance for* FEEDBACK VERTEX SET *and $H \preceq_m G$, then $(H, k)$ is a 'Yes'-instance for* FEEDBACK VERTEX SET

**Proof.** We will show that if $G$ has a feedback vertex set $S$ of size $k$ then $G', k$ has a feedback vertex set of size $k$, where $G'$ is obtained from $G$ by performing one of the three operations: (1) edge deletion, (2) deletion of an isolated vertex and (3) edge contraction.

For operations (1) and (2) the result is trivial as $S$ is a feedback vertex set in $G'$. For operation (3) , where we replace two neighboring vertices $v_1$ and $v_2$ with a vertex $v_3$, $S$ is a feedback vertex set for $G'$ unless $S$ contained $v_1$ or $v_2$, if that was the case we can add $v_3$ to $S$ to obtain a feedback vertex set of size $k$ for $G'$.

By repeated application of the above argument we can prove the result for any minor $H$ of $G$. □

We now have enough information to conclude that FEEDBACK VERTEX SET is in FPT. Although its extreme simplicity makes the technique very useful as a method for proving membership in FPT, the number of forbidden minors is often quite large and also often unknown. Thus, this technique is rarely viable as a tool for designing practical algorithms.

# 7

# LIST OF PROBLEMS

In this appendix we list all problems used in the text in alphabetical order.

## 7.1 BRANCHWIDTH

*Instance:* A graph $G = (V, E)$
*Parameter:* A positive integer $k$
*Question:* Does $G$ have branchwidth $k$?

## 7.2 CYCLE

*Input:* A graph $G = (V, E)$
*Parameter:* A positive integer $k$
*Question:* Does there exist a cycle of length $k$ in $G$?

## 7.3 DOMINATING SET

*Input:* A graph $G = (V, E)$
*Parameter:* A positive integer $k$
*Question:* Is there a set $V' \subseteq V(G)$ such that $|V'| \leq k$ and for all $v \in V$, we have $N[v] \cap V' \neq \emptyset$?

## 7.4 DOMINATING SET ON CUBIC GRAPHS

*Input:* A graph $G = (V, E)$ whose maximum degree is three
*Parameter:* A positive integer $k$
*Question:* Is there a set $V' \subseteq V(G)$ such that $|V'| \leq k$ and for all $v \in V$, $N[v] \cap V' \neq \emptyset$?

## 7.5 FEEDBACK VERTEX SET

*Input:* A graph $G = (V, E)$
*Parameter:* A positive integer $k$
*Question:* Is there a set $V' \subseteq V(G)$ such that $|V'| \leq k$ and $V - V'$ contains no cycles?

## 7.6 HAMILTONIAN CYCLE

*Input:* A graph $G = (V, E)$
*Question:* Does there exist a cycle of length $n$ in $G$?
*Comment:* Classical NP-complete problem. (Listed in [GJ79] as problem GT37.)

## 7.7 HITTING SET

*Input:* A collection $C$ of subsets of a set $S$
*Parameter:* A positive integer $k$
*Question:* Is there a set $S' \subseteq S$ such that $|S| \leq k$ and $S'$ contains at least one element from each set in $C$?

## 7.8 INDEPENDENT SET

*Input:* A graph $G = (V, E)$
*Parameter:* A positive integer $k$
*Question:* Is there a set $V' \subseteq V(G)$ such that $|V'| \geq k$ and $E(G[V']) = \emptyset$?

## 7.9 K3-PACKING

*Input:* A graph $G = (V, E)$
*Parameter:* A positive integer $k$
*Question:* Is there a collection of $k$ pairwise vertex disjoint subgraphs of $G$ each isomorphic to $K_3$?

## 7.10 MAX CUT

*I*nstance: A graph $G = (V, E)$
*P*arameter: A positive integer $k$
*Q*uestion: Is there a partitioning of $V$ into two sets $V'$,$V''$ such that the number of edges between $V'$ and $V''$ is at least k?

## 7.11 MAX LEAF SUBTREE

*I*nstance: A graph $G = (V, E)$
*P*arameter: A positive integer $k$
*Q*uestion: Does $G$ have a subtree with at least $k$ leaves?

## 7.12 MINOR ORDER TEST

*Instance:* Graphs $G = (V, E)$, $H = (V, E)$
*Parameter:* $|V(H)|$
*Question:* Is $H \preceq_m G$?

## 7.13 ODD CYCLE COVER

*Input:* A graph $G = (V, E)$
*Parameter:* A positive integer $k$
*Question:* Is there a set $V' \subseteq V(G)$ such that $|V'| \leq k$ and $V - V'$ is bipartite?

## 7.14 P2 PACKING

*Input:* A graph $G = (V, E)$
*Parameter:* A positive integer $k$
*Question:* Is there a collection of $k$ pairwise vertex disjoint subgraphs $G$ each isomorphic to $P_2$?

## 7.15 PLANAR INDEPENDENT SET

*Input:* A planar graph $G = (V, E)$
*Parameter:* A positive integer $k$
*Question:* Is there a set $V' \subseteq V(G)$ such that $|V'| \geq k$ and $E(G[V']) = \emptyset$?

## 7.16 DOMINATING SET

*Input:* A planar graph $G = (V, E)$
*Parameter:* A positive integer $k$
*Question:* Is there a set $V' \subseteq V(G)$ such that $|V'| \leq k$ and for all $v \in V$,we have $N[v] \cap V' \neq \emptyset$?

## 7.17 SHORT NONDETERMINISTIC TURING MACHINE ACCEPTANCE

*Instance:* A nondeterministic Turing machine M
*Parameter:* A positive integer $k$
*Question:* Will $M$ halt in at most $k$ steps?

## 7.18 SIZED CROWN

*Instance:* A graph $G = (V, E)$
*Parameter:* A positive integer $k$
*Question:* Does $G$ have a crown-decomposition where $|H \cup C| = k$?

## 7.19 SORTING

*Instance:* A sequence of $n$ integers $x_1, x_2, \ldots, x_n$
*Question:* What is the sequence of the same integers in increasing order?

## 7.20 TREEWIDTH

*Instance:* A graph $G = (V, E)$
*Parameter:* A positive integer $k$
*Question:* Does $G$ have treewidth $k$?

## 7.21 VERTEX COVER

*Input:* A graph $G = (V, E)$
*Parameter:* A positive integer $k$
*Question:* Is there a set $V' \subseteq V(G)$ such that $|V'| \leq k$ and $V - V'$ is an independent set?

# BIBLIOGRAPHY


[ABFKN02] J. Alber, H. L. 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.

[AFN01] J. Alber, H. Fernau, and R. Niedermeier. Parameterized complexity: exponential speedup for planar graph problems. *Proceedings of ICALP2001*, LNCS 2076, 2001

[ALS91] S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs, *Journal of Algorithms*, Vol 12, issue 2, pages 308–340, 1991.

[ALS05] F. Abu-Khzam, M. Langston, and W. Suters. Fast, Effective Vertex Cover Kernelization: A Tale of Two Algorithms, *Proceedings, ACS/IEEE International Conference on Computer Systems and Applications*, Cairo, Egypt, January, 2005.

[AYZ95] N. Alon, R. Yuster, and U. Zwick. Color-Coding, *Journal of the ACM*, Volume 42(4), pages 844–856, 1995.

[B86] H. Bodlaender. Classes of graphs with bounded treewidth, *TR RUU-CS-86-22*, Utrecht University, 1986.

[C71] S. Cook. The compexity of theorem-proving procedures, *Proceedings of 3rd annual ACM Symposium on Theory of Computing*, Association for Computing Machinery, New York, pages 151–158, 1971

[CCDF97] Liming Cai, J. Chen, R. Downey, and M. Fellows. The parameterized complexity of short computation and factorization, *Archive for Mathematical Logic* 36, pages 321–338, 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 of WG2004, LNCS*, 2004.

[CKJ01] J. Chen, I. Kanj, and W. Jia. Vertex cover: Further Observations and Further Improvements, *Journal of Algorithms* Volume 41 , pages 280–301, 2001.

[DEFPR03] R. Downey, V. Estivill-Castro, M. Fellows, E. Prieto-Rodriguez, and F. Rosamond. Cutting Up is Hard to Do: the Parameterized Complexity of $k$-Cut and Related Problems, *Electronic Notes in Theoretical Computer Science* 78, pages 205–218, 2003.

[DF99] R. Downey and M. Fellows. Parameterized Complexity, *Springer-Verlag*, 1999.

[DFRS04] F. Dehne, M. Fellows, F. Rosamond, P.Shaw. Greedy Localization, Iterative Compression and Modeled Crown Reductions: New FPT Techniques and Improved Algorithms for Max Set Splitting and Vertex Cover, *Proceedings of IWPEC04*, LNCS 3162, pages 271–281, 2004

[DFFPR05] F. Dehne, M. Fellows, H. Fernau, E. Prieto and F. Rosamond. Nonblocker: Parameterized algoritmics for Dominating Set. *to appear SOFSEM2006*

[DH05] E. Demaine and M. Hajiaghayi. Bidimensionality: New Connections between FPT Algorithms and PTASs, *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2005)*, January 23–25, pages 590–601, 2005.

[E65] J.Edmonds. Paths, trees and flowers, *Can.J.Math.*, 17, 3, pages 449–467, 1965.

[F03] M.Fellows. Blow-ups, Win/Wins and Crown Rules: Some New Directions in FPT, *Proceedings WG 2003*, Springer Verlag LNCS 2880, pages 1–12, 2003.

[FKNRSTW04] 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 12th Annual European Symposium on Algorithms (ESA 2004)*, 2004.

[FG01] J.Flum and M. Grohe. Fixed parameter tractability, definability, and model checking, *SIAM Journal on Computing* 31: pages 113–145,2001

[FG02] J. Flum and M.Grohe. Describing parameterized complexity classes. *Proceedings of 19th STACS*, LNCS 2285, pages 359–371,2002.

[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.

[FMRS00] M.R. 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, *Foundations of Software Technology and Theoretical Computer Science*, 2000.

[FT03] F. Fomin and D. Thilikos. Dominating sets in planar graphs: Branchwidth and exponentialspeed-up, *14th ACM-SIAM SODA*, pages 168–177, 2003.

[FT04] F. Fomin and D. Thilikos. A simple and fast approach for solving problems on planar graphs, *STACS*, 2004.

[G99] M.Grohe. Descriptive and Parameterized complexity, In *13th CSL* LNCS 1683, pages 14–31,1999.

[GJ79] M. Garey and D. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness, *W.H. Freeman*, San Francisco, 1979.

[H05] O. Hjortås. Branch decompositions of $k$-outerplanar graphs, *Master thesis, University of Bergen*, 2005

[K72] R.M. Karp, Reducibility among combinatorial problems', *Complexity of Computer Computations*, Plenum Press, New York, pages 85-103, 1972.

[K00] O. Kullmann. Investigations on autark assignments, *Discrete Applied Mathematics*, vol. 107, pages 99–138, 2000.

[K03] O. Kullmann. Lean clause-sets: Generalizations of minimally unsatisfiable clause-sets, *Discrete Applied Mathematics*, vol 130, pages 209–249, 2003.

[L04] M. Langston. Private communication.

[LS05] D. Lokshtanov and C. Sloper. Fixed Parameter Set Splitting, Linear Kernel and Improved Running Time,*proceedings of ACID 2005*, 2005

[L83] H. Lenstra. Integer programming with a fixed number of variables, *Mathematics of Operations Research*, 8, pages 538–548, 1983.

[M89] U. Manber. Introduction to algorithms, a creative approach, *Addison Wesley Publishing*, 1989.

[MR99] M. Mahajan, V. Raman. Parameterizing above guaranteed values: MaxSat and MaxCut, *Journal of Algorithms*, vol. 31, issue 2, pages 335-354, 1999.

[N02] R. Niedermeier. Invitation to Fixed-Parameter Algorithms. *Habilitation thesis*, unpublished, 2002

[NR00] R. Niedermeier and P. Rossmanith. A general method to speed up fixed parameter algorithms, *Information Processing Letters*, 73, pages 125–129, 2000.

[NT75] G. Nemhauser and L. Trotter Jr. Vertex Packings: Structural properties and algorithms, *Mathematical Programming*, 8, pages 232–248, 1975.

[P05] E. Prieto. Systematic kernelization in FPT algorithm design. *PhD thesis*

[P05b] E. Prieto. The Method of Extremal Structure on the $k$-Maximum Cut Problem, *Proceedings of Computing: The Australasian Theory Symposium (CATS Š05), Australian Computer Science Communications*, vol. 27(4), pages 119-Ű126, 2005.

[PS03] E. Prieto, C. Sloper. Either/Or: Using Vertex Cover Structure in designing FPT-algorithms - the case of k-Internal Spanning Tree, *Proceedings of WADS 2003*, LNCS vol 2748, pages 465–483.

[PS04] E. Prieto, C. Sloper. Looking at the Stars,*Proceedings of International Workshop on Parameterized and Exact Computation (IWPEC Š04)*, LNCS vol. 3162, pages 138-Ű149, 2004.

[PS05] E. Prieto, C. Sloper. Reducing to Independent Set Structure — the Case of $k$-INTERNAL SPANNING TREE', *Nordic Journal of Computing*, to appear.

[R01] Robson. Finding a maximum independent set in time $\mathcal{O}(2^{n/4})$? Manuscript.

[RS91] N. Robertson, P.Seymour. Graph Minors X Obstructions to tree-decomposition, *Journal Combin. Ser. B.*, 52, pages 153–190, 1991.

[RS94] N. Robertson, P. Seymour. Quickly excluding a planar graph, *Journal Combin. Ser. B.*, 62, pages 323-348, 1994.

[RS99] N. Robertson, PD. Seymor. Graph Minors XX Wagner's conjecture. *To appear*.

[RSV03] B.Reed, K.Smith, and A. Vetta. Finding Odd Cycle Transversals, *Operations Research Letters*, 32, pages 299-301, 2003.

[SS90] J.P. Schmidt and A. Siegel. The spatial complexity of oblivious $k$-probe hash functions. *SIAM Journal of Computing*, 19(5), pages 775–786, 1990.

[W03] G. Woeginger. Exact algorithms for NP-hard problems: A survey, *Combinatorial Optimization - Eureka! You shrink!*, M. Juenger, G. Reinelt and G. Rinaldi (eds.). LNCS 2570, Springer, pages 185–207, 2003.

# Part II

# Papers - case studies

# 8

# MAX INTERNAL SPANNING TREE

**Reducing to Independent Set Structure — the Case of $k$-INTERNAL SPANNING TREE**[1]

Elena Prieto Christian Sloper

**Abstract**

The $k$-INTERNAL SPANNING TREE problem asks whether a certain graph $G$ has a spanning tree with at least $k$ internal vertices. Basing our work on the results presented in [PS03], we show that there exists a set of reduction rules that modify an arbitrary spanning tree of a graph into a spanning tree with no induced edges between the leaves. Thus, the rules either produce a tree with many internal vertices, effectively deciding the problem, or they identify a large independent set, the leaves, in the graph. Having a large independent set is beneficial, because then the graph allows both 'crown decompositions' and path decompositions. We show how this crown decomposition can be used to obtain a $\mathcal{O}(k^2)$ kernel for the $k$-INTERNAL SPANNING TREE problem, improving on the $\mathcal{O}(k^3)$ kernel presented in [PS03].

## 8.1 INTRODUCTION

The subject of Parameterized Complexity is motivated by an abundance of NP-complete problems that have very different behavior when parameterized. These problems includes well-known problems like DOMINATING SET, BANDWIDTH, SET SPLITTING, and INDEPENDENT SET (for definitions the reader may refer to [GJ79]). Some of the NP-complete are tractable when parameterized and admits very good parameterized algorithms. A formal definition of the class of problems which are tractable when parameterized is defined as follows:

[1]This paper has been accepted to Nordic Journal of Computing and is due to appear.

**Definition 8.1.1** *(Fixed Parameter Tractability) A parameterized problem $L \subseteq \Sigma^* \times \Sigma^*$ is* fixed-parameter tractable *if there is an algorithm that correctly decides, in time $f(k)\, n^{\alpha}$, for input $(x, y) \in \Sigma^* \times \Sigma^*$ whether or not $(x, y) \in L$, where $n$ is the size of the input $x$, $|x| = n$, $k$ is the parameter, $\alpha$ is a constant (independent of $k$) and $f$ is an arbitrary function.*

*The class of fixed-parameter tractable problems is denoted FPT.*

It is not believed that all NP-complete problems are Fixed Parameter Tractable, the class is split into a hierarchy of classes FPT⊆W[1]⊆W[2]⊆ $\cdots$ ⊆W[P]. Here the classes W[1]⊆W[2]⊆ $\cdots$ ⊆W[P] are intractable and we justify this by a completeness-result not unlike classical complexity. In [CCDF97] Cai, Chen, Downey, and Fellows proved that $k$-SHORT NONDETERMISTIC TURING MACHINE ACCEPTANCE (Will a Nondetermistic Turing Machine halt in $k$ or less steps?) is W[1]-complete thus giving strong natural evidence that $FPT \neq W[1]$.

Further background on parameterized complexity can be found in [DF98].

The problem we address in this paper concerns spanning trees, namely $k$-INTERNAL SPANNING TREE (Does $G$ have a spanning tree with at most $n-k$ leaves?). The problem is NP-complete as HAMILTONIAN PATH can be considered a special case of $k$-INTERNAL SPANNING TREE by making $k = |V| - 2$, and HAMILTONIAN PATH is NP-complete.

In Section 8.4 we use standard techniques to show that $k$-INTERNAL SPANNING TREE is in FPT. In Section 8.5 we describe how to use the bounded *independent set structure* to design an FPT algorithm for $k$-INTERNAL SPANNING TREE. We give an analysis of the running time of the algorithm generated by the method in Section 8.6. In Section 8.7 we show how the independent structure allows a pathwidth decomposition which can be useful for some problems, we illustrate this on NONBLOCKER, the dual of DOMINATING SET. We conclude with some remarks about future research. Also, as a consequence of the preprocessing of the graph necessary to create our fixed-parameter algorithm, we easily obtain a polynomial time 2-approximation algorithm for $k$-INTERNAL SPANNING TREE.

## 8.2 USING REDUCTION RULES

Currently, the main practical methods of FPT algorithm design are based on *kernelization* and *bounded search trees*. The idea of kernelization is relatively simple, and can be quickly illustrated for the VERTEX COVER problem.

In kernelization we seek to bound the size of the input instance to a function of the parameter. To achieve this we preprocess the graph using reduction rules. Two examples of reduction rules for VERTEX COVER are the *leaf-rule* and the *Buss-rule*. The leaf-rule

states that given an instance $(G, k)$ where $G$ has a pendant vertex $v$ of degree 1 connected to vertex $u$, then it is never wrong to include $u$ in the vertex cover instead of $v$, as the edge $uv$ must be covered and $u$ possibly covers other edges as well. Thus $(G, k)$ can be reduced to $(G', k-1)$, where $G' = G - \{u, v\}$. Another rule, the Buss-rule [B98], states that if the instance $(G, k)$ has a vertex $u$ of degree greater than $k$, then $u$ must be in every $k$-vertex cover of $G$, since otherwise all its more than $k$ neighbors would have to be included. Thus, $(G, k)$ can be reduced to $(G', k-1)$ where $G' = G - u$.

The term 'Reduction rule' is somewhat unfortunate as it seems to imply a rule that reduces the graph in size. Although a reduction in size is a consequence, it is wrong to consider this the goal. Reduction rules should not be viewed as a 'reduction in size' but rather as a '*reduction to structure*'. In parameterized complexity the goal of the reduction process is to prove that the problem is after preprocessing trivially decidable for any 'large' instance, i.e., irreducible instances larger than a function $f(k)$, our kernel size. It is here that reduction rules provide us with the necessary information about the structure of the instance. In a sense, reduction rules are used to impose structure that allow us to make claims about irreducible graphs.

It is easy to be led astray by reduction rules that only offer a reduction in size, since if they do not also convey some useful structural information, then the rule is ultimately useless from the point of view of kernelization. However, such a rule could of course be very useful in practice as a preprocessing tool or in search tree algorithms.

To illustrate what we mean we again consider the leaf-rule and the Buss-rule for vertex cover. After repeated application of both we reach a graph where neither rule can be applied. We say that this graph is *irreducible* for our reduction rules. From the knowledge that the rules do not apply we can conclude that the graph has two properties. First, from the leaf-rule, we know that every vertex has degree at least 2. Second, from the Buss-rule, we know that every vertex has degree at most $k$.

Knowing that the minimum degree of the graph is at least two is important for ruling out cases in the search tree analysis, but it does not provide any 'useful' structural information as we both have arbitrarily large graphs with minimum degree at least two that have a $k$-Vertex Cover, and others that do not have a $k$-Vertex Cover. However, with the Buss-rule the situation is different. Knowing that every vertex has degree at most $k$ combined with the fact that we can select at most $k$ of them is enough to conclude that no irreducible yes-instance for $k$-Vertex Cover has more than $k(k+1)$ vertices. Thus we can trivially decide any irreducible instance of size greater than $f(k) = k(k+1)$. We have a quadratic kernel for vertex cover.

In this paper we show that we can learn something about the structure of the graph on a global level without reducing the graph in size. We show that there exists a set of reduction rules that modify an arbitrary spanning tree of a graph into a spanning tree with no induced edges between the leaves. Thus, the rules either produce a tree with many

internal vertices, effectively deciding the problem, or they identify a large independent set, the leaves, in the graph. Having a large independent set is beneficial, because then the graph allows a 'crown decomposition'. We show how this crown decomposition can be used to obtain a $\mathcal{O}(k^2)$ kernel for the $k$-INTERNAL SPANNING TREE problem, improving on the $\mathcal{O}(k^3)$ kernel presented in [PS03].

## 8.3 PRELIMINARIES

We assume simple, undirected, connected graphs $G = (V, E)$ where $|V| = n$. The set of neighbors of a vertex $v$ is denoted $N(v)$, and the neighbors of a set $S \subseteq V$ is $N(S) = \bigcup_{v \in S} N(v) - S$.

We use the simpler $G \setminus v$ to denote $G[V \setminus v]$ and $G \setminus e$ to denote $G = (V, E \setminus e)$ where $v$ and $e$ is a vertex and an edge respectively. Likewise for sets, $G \setminus V'$ denotes $G[V \setminus V']$ and $G \setminus E'$ denotes $G = (V, E \setminus E')$ where $V'$ is a set of vertices and $E'$ is a set of edges.

We say that a *$k$-internal tree* $T$ is a subgraph of $G$, where $T$ is a tree with at least $k$ internal vertices. If $V(T) = V(G)$ we say that $T$ is a *$k$-internal spanning tree* of $G$.

## 8.4 $k$-INTERNAL SPANNING TREE IS FPT

Using Robertson and Seymour's Graph Minor Theorem it is straightforward to prove the following membership in FPT.

**Lemma 8.4.1** *The* $k$-INTERNAL SPANNING TREE *problem is in* FPT.

**Proof.** Let $\mathcal{F}_k$ denote the family of graphs that do not have spanning trees with at least $k$ internal vertices. It is easy to observe that for each $k$ this family is a lower ideal in the minor order. Less formally, let $(G, k)$ be a NO-instance of $k$-INTERNAL SPANNING TREE, that is a graph $G$ for which there is no spanning tree with at least $k$ internal vertices. The local operations which configure the minor order (i.e., edge contractions, edge deletions and vertex deletions) will always transform this NO-instance into another NO-instance. By the Graph Minor Theorem of Robertson and Seymour and its companion result that order testing in the minor order is FPT [RS99] we can conclude that $k$-INTERNAL SPANNING TREE is also FPT. (An exposition of well-quasiordering as a method of FPT algorithm design can be found in [DF98].) □

Unfortunately, this FPT proof technique suffers from being nonuniform and nonconstructive, and gives an $\mathcal{O}(f(k)n^3)$ algorithm with a very fast-growing parameter function compared to the one we obtain in Section 8.5.

We remark that it can be shown that all fixed graphs with a vertex cover of size $k$ are well-quasi ordered by ordinary subgraphs and have linear time order tests [F03]. The proof of this is substantially shorter than the Graph Minor Project and could be used to simplify Lemma 8.4.1.

## 8.5 INDEPENDENT SET STRUCTURE

In this section we show how to obtain a quadratic kernel for $k$-INTERNAL SPANNING TREE. We first give a set of reduction rules that either produces a spanning tree with the desired number of internal vertices or shows that the graph has a large independent set.

We will then show that this structural information is enough to prove that any irreducible instance has size at most $\mathcal{O}(k^2)$, improving the result obtained in [PS03]. Using a *crown decomposition* we are able to prove that any graph with a large independent set contain redundant vertices that can be removed, reaching the desired kernel size.

**Lemma 8.5.1** *Any graph $G$ has a spanning tree $T$ such that all the leaves of $T$ are independent vertices in $G$ or $G$ has a spanning tree $T'$ with only two leaves.*

**Proof.** Given a spanning tree $T$ of a graph $G$, we say that two leaves $u, v \in T$ are *in conflict* if $uv \in E(G)$. We now show that given a spanning tree with $i$ conflicts it is possible to obtain a spanning tree with less than $i$ conflicts using one of the rules below:

1. If $x$ and $y$ are in conflict and $z$, the parent of $x$ has degree $3$ or more, then a new spanning tree $T'$ could be constructed using the edge $xy$ in the spanning tree instead of $xz$.

2. If $x$ and $y$ are in conflict and both their parents have degree $2$, then let $x'$ be the first vertex on a path from $x$ to $y$ that has degree different from 2. If there is no such vertex $x'$ we know that the spanning tree is a Hamiltonian path and has only two leaves. Otherwise we create a new spanning tree disconnecting the path from $x$ to $x'$ (leaving $x'$) and connecting $x$ to $y$, repairing the conflict between $x$ and $y$. Since $x'$ is now of degree at least $2$ we have not created any new conflicts.

The validity of the rules is easy to verify and it is obvious that they can be executed in polynomial time. Lemma 8.5.1 then follows by recursively applying the rules until no conflicts exist. □

Observe that any application of the rules on a spanning tree produces a spanning tree with more internal vertices, thus the reduction rules above are used less than $k$ times.

For the remainder of the paper we assume that we obtained a spanning tree $T$ where the leaves are independent and we define the set $A$ as the internal vertices of $T$ and $B$ as the leaves of $T$. Observe that $A$ is a connected set and $B$ an independent set.

Several corollaries follow easily from this Lemma. One of them gives an approximation for $k$-INTERNAL SPANNING TREE, the others relate the problem to the well-studied INDEPENDENT SET.

**Corollary 8.5.1** *$k$-INTERNAL SPANNING TREE has a 2-approximation algorithm.*

**Proof.** Note that since $B$ is an independent set it is impossible to include more than $|A|$ elements of $B$ as internals in the optimal spanning tree, as otherwise the spanning tree would contain a loop. The maximum number of internal vertices is at most $2|A|$, and since the spanning tree generated by the algorithm in Lemma 8.5.1 has $|A|$ internal vertices, it is a 2-approximation for $k$-INTERNAL SPANNING TREE. □

**Corollary 8.5.2** *If a graph $G = (V, E)$ is a* NO*-instance for $(n-k)$-*INDEPENDENT SET *then $G$ is a* YES*-instance for $k$-*INTERNAL SPANNING TREE.

**Proof.** If a graph does not have an independent set of size greater then $(n - k)$, then $|B| < (n - k)$ and $|A| \geq k$, a YES-instance of $k$-INTERNAL SPANNING TREE. □

**Corollary 8.5.3** *If a graph $G = (V, E)$ is a* YES*-instance for $(n - k)$-*INDEPENDENT SET *then $G$ is a* NO*-instance for $(2k + 1)$-*INTERNAL SPANNING TREE.

**Proof.** If $G$ has an $(n-k)$-INDEPENDENT SET $I$ then for each vertex in $I$ that we include as an internal in the spanning tree we must include at least one other vertex in $V - I$. Thus at most $2k$ vertices can be internal in the spanning tree and therefore $G$ is a NO-instance for $(2k + 1)$-INTERNAL SPANNING TREE. □

We now know that if a graph does not have an $(n - k)$-INDEPENDENT SET then it is a YES-instance for $k$-INTERNAL SPANNING TREE. We will now show how we can use this structural information to give a bound on the size of the kernel. To reduce the large independent set we will use the crown-reduction technique seen in [CFJ03, FHRST04, F03, ACFL04] to reduce the size of the independence set.

**Definition 8.5.1** *A* crown decomposition *$(H, C, R)$ in a graph $G = (V, E)$ is a partitioning of the vertices of the graph into three sets $H$, $C$, and $R$ that have the following properties:*

1. $H$ (the head) *is a separator in* $G$ *such that there are no edges in* $G$ *between vertices in* $C$ *and vertices in* $R$.
2. $C = C_u \cup C_m$ (the crown) *is an independent set in* $G$.
3. $|C_m| = |H|$, *and there is a perfect matching between* $C_m$ *and* $H$.

Although being a recently introduced idea, some theory about the existence of crowns can be found in literature.

The following theorem can be deduced from [CFJ03, page 7], and [F03, page 8].

**Theorem 8.5.1** *Any graph* $G$ *with an independent set* $I$, *where* $|I| \geq n/2$, *has a crown decomposition* $(H, C, R)$, *where* $H \subseteq N(I)$ *and* $C \subseteq I$, *and this crown decomposition can be found in time* $\mathcal{O}(|V| + |E|)$, *given* $I$.

In [FHRST04] the following is observed:

**Lemma 8.5.2** *If a bipartite graph* $G = (V \cup V', E)$ *has two crown decompositions* $(H, C, R)$ *and* $(H', C', R')$ *where* $H \subseteq V$ *and* $H' \subseteq V$, *then* $G$ *has a crown decomposition* $(H'' = H \cup H', C'' = C \cup C', R'' = R \cap R')$.

From these two results we can deduce that if the independent set is sufficiently large then there exists a crown-decomposition where $C_u \neq \emptyset$.

**Theorem 8.5.2** *Any graph* $G$ *with an independent set* $I$, *where* $|I| \geq 2n/3$, *has a crown decomposition* $(H, C, R)$, *where* $H \subseteq N(I)$, $C \subseteq I$ *and* $C_u \neq \emptyset$, *that can be found in time* $\mathcal{O}(|V||E|)$ *given* $I$.

**Proof.** First observe that $|C_m| \leq |N(I)|$. By Theorem 8.5.1, $G$ has a crown decomposition $(H, C, R)$, where $H \subseteq N(I)$. If $|C| \geq \frac{n}{3}$ then $|C| > N(I)$ and the result follows, otherwise $|I \setminus C| \geq n/3$ and by Theorem 8.5.1 $G \setminus C$ has a crown decomposition $(H'', C', R')$. By Lemma 8.5.2 these crown-decompositions can be combined to a crown-decomposition $(H'', C'', R'')$. This process can be repeated until the combined crown-decomposition $(\hat{H}, \hat{C}, \hat{R})$ no longer satisfies $|I \setminus \hat{C}| > \frac{n}{3}$, thus $|\hat{C}| > |N(I)|$ and the result follows. The algorithm in Theorem 8.5.1 is executed at most $n$ times, giving the bound of $\mathcal{O}(|V||E|)$. □

Using an approach similar to the one in [FHRST04], we create an auxiliary graph model where a crown decomposition in the auxiliary graph infer reductions in the original graph.

Observe that vertices in the independent set $B$ can only participate in a spanning tree in two ways. Either they are leaves, or they are internal vertices between two or more vertices in $A$.

We will define the model as the bipartite graph $G_I = (A' \cup B, E_I)$ where: $A' = A \cup (A \times A)$, i.e., $A$ and a vertex $vv'$ for every pair $v$ and $v'$ in $A$. The edges of $G_I$ are the original edges $E$ and an edge between a vertex $b \in B$ and a pair vertex if $b$ has edges to both vertices of the pair. $E_I = E \cup \{(vv')b \mid vv' \in A', b \in B, \{vb, v'b\} \subseteq E\}$.

We now prove the following reduction rule.

**Reduction Rule 3** *If $G_I$ has a crown decomposition $(H, C_m \cup C_u, R)$ where $H \subseteq A'$ then $G$ has a $k$-internal spanning tree if and only if $G \setminus C_u$ has a $k$-internal spanning tree.*

**Proof.** One direction is trivial, if $G \setminus C_u$ has a $k$-internal spanning tree then $G$ obviously has one, as we cannot get fewer internals by adding vertices to the graph.

We prove the other direction by construction. Let $S^*$ be a $k$-internal spanning tree in $G$. $S^* - C$ is a forest $F$. We will show that we can construct a $k$-internal spanning tree from $F$ by using vertices from $C_m$ to connect the components in $F$, showing that $C_u$ is redundant.

Let $Q$ be the components of $F$. Observe that at most $|Q| - 1$ vertices from $C$ connected the components in $S^*$ and that all these vertices are internal.

Let $Q_i$ and $Q_j$ be two arbitrary components in $Q$ that were connected by a vertex $c \in C$. Let $u_i$ and $u_j$ be the vertices in $Q_i$ and $Q_j$ respectively of which $c$ is a neighbor in $S^*$. Connect these vertices using the vertex in $C_m$ matched to the pair-vertex $u_iu_j$. Because of the matching in the crown decomposition, this vertex is uniquely determined and never used elsewhere in the construction. The number of components have decreased by one. Repeat this process until all components in $Q$ are connected. Note that we added $|Q| - 1$ internal vertices, thus we used at least as many vertices to connect $F$ as the optimal solution did. $F$ is now a tree.

For every leaf $u_i$ in $F$ which is not a leaf in $S^*$, append the vertex matched to $u_i \in C_m$. As above, the vertex matched to $u_i$ is uniquely determined and not used elsewhere in the construction.

Note that the construction of the $k$-internal spanning tree never depends on $C_u$, thus $C_u$ is redundant. □

**Lemma 8.5.3** *If $G$ is reduced and $|V(G)| > k^2 + 2k$ then $G$ has a $k$-Internal Spanning Tree.*

**Proof.** Assume in contradiction to the stated lemma that $G$ is reduced and $|V(G)| > k^2 + 2k$, but that $G$ has no $k$-Internal Spanning Tree.

By assumption $|A| < k$, otherwise the tree produced in Lemma 8.5.1 would have $k$ internal vertices. Hence, $|B| = |V(G) - A| > k^2 + k$. In $G_I$ we have that is $|A'| < k(k+1)/2$, i.e., $|B| > 2|A'|$. Thus by Lemma 9.4.3, $G_I$ has a crown with at least one vertex in $C_u$, contradicting the assumption that $G$ was reduced.

□

## 8.6 ANALYSIS OF THE RUNNING TIME

Our algorithm is similar to that found in [PS03] and works in several stages. It first calls a regular spanning tree algorithm and then modifies it to make the leaves independent. Then, if the spanning tree does not contain enough internals, we know that the spanning tree's leaves form an independent set. We use our crown reduction rule to reduce the independent set, after which the graph is reduced in size to $\mathcal{O}(k^2)$. Finally, we employ a brute-force spanning tree algorithm to find an optimal solution for the reduced instance.

We can use a simple breadth-first search algorithm to obtain any spanning tree in $G$. This spanning tree can thus be obtained in time $\mathcal{O}(|V| + |E|)$ [CLR90]. The conflicts (i.e., the leaves in the tree which are not independent) can be detected in time $\mathcal{O}(|E|)$ and repaired in time $\mathcal{O}(|V|)$.

Given a large independent set, a crown can be found in linear time. A maximal crown can be found in time $\mathcal{O}(|V||E|)$. We have then identified the redundant vertices and we can reduce the graph to a $\mathcal{O}(k^2)$ kernel.

We now want to find $k$ vertices in the kernel that can form the internals of a spanning tree. We will in a brute force manner test every such $k$-set, there are at most $\binom{k^2}{k}$ such sets. By Stirling's observation that $n^{\frac{n}{2}} < n! < n^n$ we have that $\binom{k^2}{k}$ is less than $k^{\frac{3}{2}k}$. Note that this can be rewritten as $2^{1.5k \log k}$. We now have to verify if these $k$ vertices can be used as the internal vertices of a spanning tree. To do this we try every possible construction of a tree $T$ with these $k$ vertices, by Cayley's formula there are no more than $k^{k-2}$ such trees. This, again, can be rewritten as $(2^{k \log k - 2 \log k})$. Then we test whether or not each leaf in $T$ can be assigned at least one vertex in the remaining kernel as its leaf. This is equivalent to testing if the leaves and the remaining kernel have a perfect bipartite matching, which can be done in time $\mathcal{O}(\sqrt{|V|} \cdot |E|)$. In this particular bipartite subset there are not more than $\mathcal{O}(k^3)$ edges giving us a total of $\mathcal{O}(k^4)$ for the matching. Thus for each $k$-set we can verify if it is a valid solution in $2^{k \log k} \cdot k^2$ time.

The total running time of the algorithm is $\mathcal{O}(2^{2.5k \log k} k^2 + |V||E|)$.

## 8.7 ANOTHER PATH(WIDTH) TO SUCCESS

If we cannot use crown-decompositions to reduce the graph efficiently, we can sometimes make use of the fact that the independent structure allows an easy path-decomposition as well. The notion of pathwidth was introduced by Robertson and Seymour [RS83].

**Definition 8.7.1** *A* path decomposition *of a graph $G = (V, E)$ is a sequence $(X_1, X_2, \ldots, X_r)$ of subsets of $V$ such that:*

1. $\bigcup_{1 \leq i \leq r} X_i = V$.
2. *For all $vw \in E$, there is an $i$ such that $1 \leq i \leq r$ and $v, w \in X_i$.*
3. *For all $1 \leq i_0 \leq i_1 \leq i_2 \leq r$, we have $X_{i_0} \cap X_{i_2} \subseteq X_{i_1}$.*

*The* width *of a path decomposition $(X_1, X_2, \ldots, X_r)$ is $\max_{1 \leq i \leq r} |X_i| - 1$. The* pathwidth *of a graph is the minimum width over its path decompositions.*

If we have an independent set $I$ of size $n - g(k)$ we can create a path decomposition with width $g(k)$ in the following manner. Let $I_1, I_2, \ldots$ be an arbitrary ordering of $I$. The path decomposition is then the sequence of subsets $B_j = \overline{I} \cup I_j$. It is easy to convince oneself that this construction satisfies the requirements of a path decomposition.

To give an example where this is useful, consider the parametric dual of $k$-DOMINATING SET, namely $k$-NONBLOCKER. (Does $G = (V, E)$ have a subset $V'$ of size $k$, such that every element of $V'$ has at least one neighbor in $V \setminus V'$?).

**Lemma 8.7.1** *$k$-NONBLOCKER can be solved in time $\mathcal{O}(3^k + n^{\mathcal{O}(1)})$.*

To show this observation, we first compute a *maximal* independent set I. The complement of $I$, $\overline{I} = V \setminus I$ is a nonblocking set. Thus either $|\overline{I}| < k$ or $G$ has a $k$-NONBLOCKER. We can then compute a path decomposition with pathwidth $k$. Now, using the algorithm introduced by Telle and Proskurowski [PT93] and further improved by Alber and Niedermeier [AN02] we can compute a minimum dominating set (and thus maximal nonblocking set) in time $\mathcal{O}(3^k + n^\alpha)$. The above algorithm actually solves the problem for the more general treewidth decomposition in time $\mathcal{O}(4^k + n^\alpha)$, but since this is a path decomposition we can avoid the costly functions combining subtrees of the decompositions. This result improves on the running time of McCartin's algorithm [McC03], which obtains a $\mathcal{O}(4^k + n^\alpha)$ algorithm by using a very different technique.

## 8.8 CONCLUSIONS AND FURTHER APPLICATIONS TO INDEPENDENT SET STRUCTURES

In this paper we have given a fixed parameter algorithm for $k$-INTERNAL SPANNING TREE. The algorithm runs in time $\mathcal{O}(2^{2.5k \log k} \cdot k^2 + |V||E|)$, which is the best currently known for this problem. A natural question is whether or not the there is a $2^{\mathcal{O}(k)}$ algorithm for the problem.

We also give a 2-approximation algorithm for the problem. This could be further improved, and the same idea could be used to find more approximation algorithms for other related problems. We would like to note that a limited number of experiments suggest that this algorithm is a very good heuristic.

We have shown the remarkable structural bindings between $k$-INTERNAL SPANNING TREE and $(n-k)$-INDEPENDENT SET in Corollaries 8.5.2 and 8.5.3. We believe that similar structural bindings exist between INDEPENDENT SET/VERTEX COVER ($k$-Vertex Cover is of course equivalent to $(n-k)$-INDEPENDENT SET) and other fixed-parameter tractable problems. We are confident that this inherent structure can be used to design potent algorithms for these problems, especially when combined with constructive polynomial time algorithms that produce either an independent set or a solution for the problem in question. Crown decompositions seem to be a natural companion as it has shown itself useful in reducing independent sets in a range of problem [FHRST04, PS04, MPS04, DFRS04, CFJ04].

We also show how the independent set structure allows an easy path decomposition and show that this is useful for $k$-NONBLOCKER where we improve upon the existing FPT-algorithms.

If large independent sets are the targets, but no such polynomial *either/or* algorithm can be found, we may still use the quite practical FPT VERTEX COVER-algorithm to find the vertex cover structure. The current state of the art algorithm for VERTEX COVER runs in time $\mathcal{O}(1.286^k + n)$ [CKJ01] and has been proven useful in implementations by groups at Carleton University in Ottawa and the University of Tennessee in Knoxville for exact solutions for values of $n$ and $k$ up to 2,500 [L03]. We believe that exploiting vertex cover/independent set structure may be a powerful tool for designing algorithms for other fixed parameter tractable problems for which structural bindings with INDEPENDENT SET exist. For example, we suspect that the parameterized versions of MAX LEAF SPANNING TREE, MINIMUM INDEPENDENT DOMINATING SET and MINIMUM PERFECT CODE are very likely to fall into this class of problems.

# BIBLIOGRAPHY


[A03] F. Abu-Khzam. Private communication.

[ACFL04] F. Abu-Khzam, R. Collins, M. Fellows and M. Langston. Kernelization Algorithms for the Vertex Cover Problem: Theory and Experiments. *Proceedings ALENEX 2004*, Springer-Verlag, *Lecture Notes in Computer Science* (2004), to appear.

[AN02] J. Alber and R. Niedermeier. Improved tree decomposition based algorithms for domination-like problems. *Proceedings of the 5th Latin American Theoretical INformatics (LATIN 2002)*, number 2286 in Lecture Notes in Computer Science, pages 613–627, Springer (2002).

[B98] S. Buss, *listed as private communication in the book Parameterized Complexity*

[CFJ03] B. Chor, M. Fellows, D. Juedes. Private communication concerning manuscript in preparation.

[CFJ04] B. Chor, M. Fellows, D. Juedes. Linear Kernels in Linear Time, or How to Save k Colors in $\mathcal{O}(n^2)$ steps. To appear in proceedings 30th Workshop on Graph Theoretic Concepts in Computer Science (WG '04), Springer Lecture Notes in Computer Science, (2004).

[CCDF97] Liming Cai, J. Chen, R. Downey and M. Fellows. The parameterized complexity of short computation and factorization. *Archive for Mathematical Logic* 36 (1997), 321-338.

[CKJ01] J. Chen, I. Kanj, and W. Jia. Vertex cover: Further Observations and Further Improvements. *Journal of Algorithms* Volume 41, 280-301 (2001).

[CLR90] T.H.Cormen, C.E.Leierson, R.L.Rivest, *Introduction to Algorithms*, MIT Press.

[DF98] R. Downey and M. Fellows. *Parameterized Complexity* Springer-Verlag (1998).

[DFRS04] F. Dehne, M. Fellows, F. Rosamond, P.Shaw. Greedy Localization, Iterative Compression and Modeled Crown Reductions: New FPT Techniques and Improved Algorithms for Max Set Splitting and Vertex Cover. *To Appear at IWPEC04* Springer Lecture Notes in Computer Science, (2004).

[DFS99] R. Downey, M. Fellows and U. Stege. Parameterized complexity: a framework for systematically confronting computational intractability. *Contemporary Trends in Discrete Mathematics* (R. Graham, J. Kratochvil, J. Nesetril and F. Roberts, eds.), *AMS-DIMACS Series in Discrete Mathematics and Theoretical Computer Science* 49 (1999), 49-99.

[F03] M.Fellows. Blow-ups, Win/Wins and Crown Rules: Some New Directions in *FPT*. *Proceedings WG 2003*, Springer Verlag LNCS 2880, pages 1-12, 2003.

[FHRST04] M.Fellows, P.Heggernes, F.Rosamond, C. Sloper, J.A.Telle, Finding k disjoint triangles in an arbitrary graph. To appear in proceedings *30th Workshop on Graph Theoretic Concepts in Computer Science (WG '04)*, Springer Lecture Notes in Computer Science, (2004).

[FMRS01] M. Fellows, C. McCartin. F. Rosamond and U. Stege. Spanning Trees with Few and Many Leaves. *To appear*

[GMM94] G. Galbiati, F. Maffioli, and A. Morzenti. A Short Note on the Approximability of the Maximum Leaves Spanning Tree Problem. *Information Processing Letters* 52 (1994), 45–49.

[GMM97] G. Galbiati, A. Morzenti and F. Maffioli. On the Approximability of some Maximum Spanning Tree Problems. *Theoretical Computer Science* 181 (1997), 107–118.

[GJ79] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*.W.H. Freeman, San Francisco, 1979.

[KR00] Subhash Khot and Venkatesh Raman. *Parameterized Complexity of Finding Hereditary Properties*. Proceedings of COCOON. Theoretical Computer Science (COCOON 2000 special issue)

[L03] M. Langston. Private communication.

[LR98] H.-I. Lu and R. Ravi. Approximating Maximum Leaf Spanning Trees in Almost Linear Time. *Journal of Algorithms* 29 (1998), 132–141.

[MPS04] L. Mathieson, E. Prieto, P. Shaw. Packing Edge Disjoint Triangles: A Parameterized View. *To Appear IWPEC 04*, Springer Lecture Notes in Computer Science, (2004).

[McC03] Catherine McCartin. Ph.D. dissertation in Computer Science, Victoria University, Wellington, New Zealand, (2003).

[NR99b] R. Niedermeier and P. Rossmanith. Upper Bounds for Vertex Cover Further Improved. In C. Meinel and S. Tison, editors, *Proceedings of the 16th Symposium on Theoretical Aspects of Computer Science*, number 1563 in Lecture Notes in Computer Science, Springer-Verlag (1999), 561–570.

[PS03] E. Prieto, C. Sloper. Either/Or: Using Vertex Cover Structure in designing FPT-algorithms - the case of k-Internal Spanning Tree, *Proceedings of WADS 2003*, LNCS vol 2748, pp 465-483.

[PS04] E. Prieto, C. Sloper. Looking at the Stars, *To Appear in proceedings of IWPEC04*, Springer Lecture Notes in Computer Science, (2004).

[PT93] J.A.Telle and A.Proskurowski. Practical algorithms on partial $k$-trees with an application to domination-like problems. *Proceedings WADS'93 - Third Workshop on Algorithms and Data Structures.* Springer Verlag, Lecture Notes in Computer Science vol.709 (1993) 610-621.

[RS83] N. Robertson and P. D. Seymour, Graph minors. I. Excluding a forest, *J. Comb. Theory Series B*, 35 (1983), pp. 39-61.

[RS99] N. Robertson, PD. Seymor. Graph Minors. XX Wagner's conjecture. *To appear*.

# 9

# PACKING STARS

**Looking at the Stars**[1]

Elena Prieto Christian Sloper

**Abstract**

The problem of packing $k$ vertex-disjoint copies of a graph $H$ into another graph $G$ is NP-complete if $H$ has more than two vertices in some connected component. In the framework of parameterized complexity we analyze a particular family of instances of this problem, namely the packing of stars. We give a quadratic kernel for packing $k$ copies of $H = K_{1,s}$. When we consider the special case of $s = 2$, i.e., $H$ being a star with two leaves, we give a linear kernel and an algorithm running in time $\mathcal{O}(2^{5.301k}k^{2.5} + n^3)$.

## 9.1 INTRODUCTION

The problem of MAXIMUM $H$-MATCHING, also called MAXIMUM $H$-PACKING, is of practical interest in the areas of scheduling [BM02], wireless sensor tracking [BK01], wiring-board design and code optimization [HK78] and many others.

The problem is defined as follows: Let $G = (V, E)$ be a graph and $H = (V_H, E_H)$ be a fixed graph with at least three vertices in some connected component. An $H$-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 would be to find the maximum number of vertex disjoint copies of $H$ in $G$. The problem is NP-complete [HK78] when the graph $H$ has at least three vertices in some connected component. Note that in the case where

[1]This paper first appeared at the conference 'First International Workshop on Parameterized And Exact Computation' in September 2004. It was later invited and accepted to a special use of the Journal of Theoretical Computer Science where it is due to appear [PS04].

$H$ 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 MAX-SNP-complete [K94] and approximable within $|V_H|/2 + \varepsilon$ for any $\varepsilon > 0$ [HS89]. Several restrictions have also been considered (planar graphs, unit disc graphs etc.) in terms of the complexity of their approximation algorithms. For a review of these we refer the reader to [AC99].

A recent result by [FHRST04] gives a general algorithm for packing an arbitrary graph $H$ into $G$. Their result gives a $2^{\mathcal{O}(|H|k \log k + k|H| \log |H|)}$ algorithm for the general case, where $k$ is the number of copies of $H$. It should also be noted that it is possible to achieve a single exponential running time for this problem by adapting a result by Alon, Yuster, and Zwick in [AYZ95].

**Theorem 9.1.1** *(Alon, Yuster, Zwick) Let $S$ be a directed or undirected graph on $k$ vertices with treewidth $t$. Let $G = (V, E)$ be a (directed or undirected) graph. A subgraph of $G$ isomorphic to $S$, if one exists, can be found in $2^{\mathcal{O}(k)}|V|^{t+1}$ expected time and in $2^{\mathcal{O}(k)}|V|^{t+1} \log |V|$ worst case time.*

It is easy to see how to apply this problem to packing a graph $H$. Let the graph $S$ in the above theorem be $k$ copies of a graph $H$. Since $S$ has treewidth at most $|H|$, we have a $2^{\mathcal{O}(k)}|V|^{|H|+1}$ algorithm for the problem. Unfortunately the running time obtained by Alon *et al.* [AYZ95] hides a considerable constant in the exponent making this algorithm infeasible in practical terms.

We discuss the parameterized complexity of the MAXIMUM $H$-PACKING problem for the case when $H$ belongs to the restricted family of graphs $\mathcal{F} = K_{1,s}$, a star with $s$ leaves. More formally:

> $K_{1,s}$-PACKING
> INSTANCE: Graph $G = (V, E)$, a positive integer $k$
> QUESTION: Are there at least $k$ vertex disjoint instances of $K_{1,s}$ in $G$?

This problem has already been studied within the framework of classical complexity theory [HK86]. In their paper, Hell and Kirkpatrick studied the complexity of packing complete bipartite graphs into general graphs. We include a brief introduction to this topic in Section 9.2. In Section 9.3 we show that the general problem is tractable if parameterized, and that we can obtain a quadratic kernel. In Section 9.4 we show that the special case of packing $K_{1,2}$'s has a linear kernel, and in Section 9.5 we give a quick algorithm for both the general and special case. In contrast [FHRST04] obtains only an $\mathcal{O}(k^3)$ algorithm for packing a graph with three vertices, namely $K_3$.

## 9.2 INTRODUCTION TO PARAMETERIZED ALGORITHMS

A problem with main input $x$ and parameter $k$ is said to be fixed parameter tractable if there is an algorithm with running time $\mathcal{O}(f(k)|x|^{\mathcal{O}(1)})$, where $f$ is an arbitrary function. In [F03] Mike Fellows presents a two-sided view of research on parameterized problems which he dub 'the two races'. First, that it is interesting to obtain better running time for fixed parameter tractable problems, but also that is of interest to improve the size of the *kernel* even if this does not immediately lead to an improvement in running time.

**Definition 9.2.1** *A parameterized problem $L$ is kernelizable if there is a parametric transformation of $L$ to itself that satisfies:*

1. *The running time of the transformation of $(x, k)$ into $(x', k')$, where $|x| = n$, is bounded by a polynomial $q(n, k)$,*
2. *$k' \leq k$, and*
3. *$|x'| \leq h(k')$, where $h$ is an arbitrary function.*

Obviously the two views are not independent, as improvements in the latter could give improvements in the first, but it is also important to note the following result by [DFS99], which gives a stronger link between the two races:

**Lemma 9.2.1** *A parameterized problem $L$ is in FPT if and only if it is kernelizable.*

The two races are worth playing as they may lead to substantial improvements on the quality of the algorithms we design and also to new strategies for practical implementations of these algorithms.

### 9.2.1 PRELIMINARIES

We assume simple, undirected, connected graphs $G = (V, E)$ where $|V| = n$. The set of neighbors of a vertex $v$ is denoted $N(v)$, and the neighbors of a set $S \subseteq V$, $N(S) = \bigcup_{v \in S} N(v) \setminus S$. If $J$ is a collection of graphs, then $V(J)$ is the set of vertices in the graphs in $J$.

The induced subgraph of $S \subseteq V$ is denoted $G[S]$.

We use the simpler $G \setminus v$ to denote $G[V \setminus \{v\}]$ for a vertex $v$ and $G \setminus e$ to denote $G = (V, E \setminus \{e\})$ for an edge $e$. Likewise $G \setminus V'$ denotes $G[V \setminus V']$ and $G \setminus E'$ denotes $G = (V, E \setminus E')$ where $V'$ is a set of vertices and $E'$ is a set of edges.

We say that $K_{1,s}$ is an $s$-star or a star of size $s$. $P_i$ denotes a path of $i + 1$ vertices and $i$ edges.

## 9.3 PARAMETERIZED COMPLEXITY OF STAR PACKING

In this section we prove a series of polynomial time preprocessing rules (reduction rules) and eventually show that we can obtain a kernel of $\mathcal{O}(k^2)$ vertices for the parameterized version of $K_{1,s}$-packing.

We use the following natural parameterization of $K_{1,s}$-PACKING:

$k$-$K_{1,s}$-PACKING
INSTANCE: Graph $G = (V, E)$
PARAMETER: $k$
QUESTION: Are there $k$ vertex disjoint instances of $K_{1,s}$ in $G$?

In order to remove vertices of high degree, and remove useless edges between vertices of low degree, we introduce the following reduction rules.

**Lemma 9.3.1** *Let $G$ be a graph with a vertex $v$ where deg$(v) > k(s+1) - 1$. Then $G$ has a $k$-$K_{1,s}$-packing if and only if $G \setminus v$ has a $(k-1)$-$K_{1,s}$-packing.*

**Proof.** If $G$ has a $k$-$K_{1,s}$-packing, then $G'$ obviously has a $(k-1)$-$K_{1,s}$, as $v$ cannot participate in two different stars.

If $G'$ has a $(k-1)$-$K_{1,s}$-packing, we can create a $k$-$K_{1,s}$-packing 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. □

**Lemma 9.3.2** *Let $G$ be a graph with neighboring vertices $u$ and $v$ where deg$(u) \leq$ deg$(v) < s$. Then $G$ has a $k$-packing if and only if $G' = (V, E(G) \setminus uv)$ contains a $k$-packing.*

**Proof.** If $G$ has a $k$-$K_{1,s}$-packing, then $G'$ has a $k$-$K_{1,s}$-packing, as $uv$ can never participate in a $K_{1,s}$. The other direction is trivial, if $G'$ has a $k$-$K_{1,s}$-packing, then $G$ has a $k$-$K_{1,s}$-packing as well. □

In order to give a quadratic kernel for the fixed parameter version of $k$-STAR PACKING we will use a new technique first seen in [FM+00]. This technique borrows ideas from extremal graph theory. We will show that any graph where Lemmas 9.3.1 and 9.3.2 do not apply is either 'small' (having less than $g(k)$ vertices) or has a $k$-$K_{1,s}$-PACKING. We do this by studying a 'border'-line graph $G$: A graph with a $k$-$K_{1,s}$-packing, but no $(k+1)$-$K_{1,s}$-packing. This allows us to make claims about the structure of $G$ and finally to prove a bound on $|V(G)|$.

A graph is *reduced* when Lemmas 9.3.1 and 9.3.2 can not be applied. In this sense both these lemmas will be commonly referred to as *reduction rules*. As an additional reduction rule, we delete vertices of degree 0, as they never participate in any star.

**Lemma 9.3.3** *(Boundary Lemma) If a graph instance $(G, k)$ is reduced and has a $k$-$K_{1,s}$-packing, but no $(k+1)$-$K_{1,s}$-packing then $|V(G)| \leq k(s^3 + ks^2 + ks + 1)$.*

**Proof.** Assume there exists a counterexample $G$, such that $G$ is reduced and contains a $k$-$K_{1,s}$-packing $W$, but no $(k+1)$-$K_{1,s}$-packing and has size $|V(G)| > k(s^3+ks^2+ks+1)$.

Let $Q = V \setminus W$. Let $Q_i$ be the vertices in $Q$ that have degree $i$ in the subgraph induced by $Q$. We will now prove a series of claims that bound the number of vertices in $Q$.

**Claim 1** $\forall i \geq s, Q_i = \emptyset$

**Claim 2** *A $K_{1,s}$-star $S \in W$ has at most $s^2 + k(s+1) - 1$ neighbors in $Q$.*

The following claim follows from Claim 2:

**Claim 3** *$W$ has at most $k(s^2 + k(s+1) - 1)$ neighbors in $Q$.*

Let $R = V \setminus (W \cup N(W))$, i.e., the set of vertices of $Q$ which do not have neighbors in $W$.

**Claim 4** *$R$ is an independent set in $G$.*

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

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

From this boundary lemma follows that any reduced instance that is still 'big' has a $k$-$K_{1,s}$-packing. Since the boundary given by Lemma 9.3.3 does not depend on the main input, but only on the parameter and the problem in question, we can say that the reduced instance is a 'problem-kernel' and that the problem is in FPT.

**Lemma 9.3.4** *(Kernelization Lemma) If a graph $G$ is reduced and has $|V(G)| > k(s^3 + ks^2 + ks + 1)$, then it contains a $k$-$K_{1,s}$-packing.*

**Proof.** Assume in contradiction to the stated theorem that there exists a graph $G$ of size $|V(G)| > k(s^3 + ks^2 + ks + 1)$, having no $k$-$K_{1,s}$-packing.

Let $k' < k$ be the largest $k'$ for which $G$ is a YES-instance. By the Boundary Lemma 9.3.3, we know that $|V(G)| \leq k'(s^3 + k's^2 + k's + 1) < k(s^3 + ks^2 + ks + 1)$. This contradicts the assumption. □

Thus for any $k$-$K_{1,s}$-packing we can prove a quadratic kernel. However, for the special case $s = 2$, we can improve on this. This is the topic of the next section.

## 9.4 THE SPECIAL CASE OF $\mathbf{P_2}$: A LINEAR KERNEL

A 2-star can also be seen as a path with three vertices, denoted $P_2$. For this special case we can employ a different set of reduction rules to obtain a linear kernel for packing $P_2$'s into a graph.

$k$-$P_2$-PACKING
INSTANCE: Graph $G = (V, E)$
PARAMETER: $k$
QUESTION: Are there $k$ vertex disjoint instances of $P_2$ in $G$?

To improve on the quadratic kernel obtained in the previous section, we will make use of a series of reduction rules based on the ideas of crown decompositions [CFJ03].

**Definition 9.4.1** *A* crown decomposition $(H, C, R)$ *in a graph* $G = (V, E)$ *is a partitioning of the vertices of the graph into three sets* $H$*,* $C$*, and* $R$ *that have 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. $C = C_u \cup C_m$ (the crown) *is an independent set in* $G$*.*
3. $|C_m| = |H|$*, and there is a perfect matching between* $C_m$ *and* $H$*.*

There are several recent papers that use crown decompositions of graphs to obtain good results in parameterized complexity [CFJ03, FHRST04, F03, ACFL04, PS04]. These papers either apply the crown directly to the problem instance ([CFJ03, ACFL04]) or create an auxiliary graph where they apply crown reduction techniques.

In this paper we instead modify the crown decomposition to fit our particular problem. The first variation is *double crown decomposition* where each vertex in $H$ has two vertices from $C$ matched to it (as opposed to only one). See Figure 9.1.

**Definition 9.4.2** *A* double crown decomposition $(H, C, R)$ *in a graph* $G = (V, E)$ *is a partitioning of the vertices of the graph into three sets* $H$*,* $C$*, and* $R$ *that have the following properties:*

1. $H$ (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. $C = C_u \cup C_m \cup C_{m2}$ (the crown) *is an independent set in* $G$*.*
3. $|C_m| = |H|$*,* $|C_{m_2}| = |H|$ *and there is a perfect matching between* $C_m$ *and* $H$*, and a perfect matching between* $C_{m_2}$ *and* $H$*.*

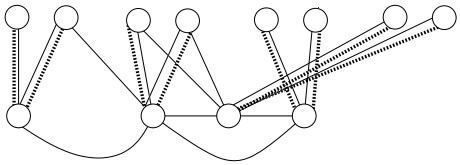

Figure 9.1: Example of 'double crown'. The dashed lines indicate how each vertex in $H$ is matched to two vertices in $C$.

Another variation of the crown is the *fat crown decomposition* where instead of independent vertices in $C$ we have $K_2$'s as shown in Figure 9.2.

**Definition 9.4.3** *A* fat crown decomposition $(H, C, R)$ *in a graph* $G = (V, E)$ *is a partitioning of the vertices of the graph into three sets* $H$*,* $C$ *and* $R$ *that have 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. $G[C]$ *is a forest where each component is isomorphic to* $K_2$*.*
3. $|C| \geq |H|$*, and if we contract the edges in each* $K_2$ *there is a perfect matching between* $C$ *and* $H$*.*

Using the 'crown', 'double crown' and 'fat crown' we can create powerful reduction rules.

**Lemma 9.4.1** *A graph* $G = (V, E)$ *that admits a 'double crown'-decomposition* $(H, C, R)$ *has a* $k$*-*$P_2$*-packing if and only if* $G \setminus (H \cup C)$ *has a* $(k - |H|)$*-*$P_2$*-packing.*

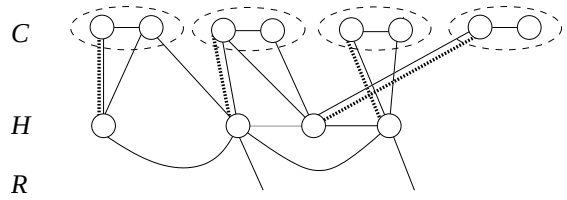


Figure 9.2: Example of 'fat crown'. As in the case of the 'double crown', the dashed lines indicate the matching between $H$ and $C_m$ and the dashed ellipses show to which $K_2$ the vertex in $H$ is matched.

**Proof.**

($\Leftarrow$:) If $G \setminus (H \cup C)$ has a $(k - |H|)$-$P_2$-packing then it is obvious that $G$ has a $k$-$P_2$-packing as $H \cup C$ has a $|H|$-$P_2$-packing ($v \in H$ and $v$'s matched vertices from $C_m$ and $C_{m_2}$ form a $P_2$).

($\Rightarrow$:) We want to prove that if $G$ has a $k$-$P_2$-packing then $G \setminus (H \cup C)$ has a $(k - |H|)$-$P_2$-packing. Assume in contradiction that there exists a graph $G'$ that has a crown-decomposition $(H', C', R')$ that contradicts the lemma. This implies that $H' \cup C'$ participates in $x > |H'|$ different $P_2$'s. Since $H'$ is a cutset, and $C$ is an independent set in the graph, every $P_2$ in $G$ that has vertices in $H' \cup C'$ must contain at least one vertex of $H'$. Thus we can have at most $|H'|$ different $P_2$'s which is a contradiction. □

**Lemma 9.4.2** *A graph $G = (V, E)$ that admits a 'fat crown'-decomposition $(H, C, R)$ has a $k$-$P_2$-packing if and only if $G \setminus (H \cup C)$ has a $(k - |H|)$-$P_2$-packing.*

The proof of Lemma 9.4.2 is analog to that of Lemma 9.4.1, thus omitted.

To apply crown-decompositions we need to know when we can expect to find one. A very useful result in this regard can be deducted from [CFJ03, page 7], and [F03, page 8]. Fortunately, the results also apply to the variations of crown decomposition described here.

**Lemma 9.4.3** *Any graph $G$ with an independent set $I$, where $|I| \geq |N(I)|$, has a crown decomposition $(H, C, R)$, where $H \subseteq N(I)$ that can be found in $\mathcal{O}(|V| + |E|)$ time given $I$.*

**Corollary 9.4.1** *Any graph $G$ with a collection $J$ of independent $K_2$'s where $|N(V(J))| \leq |J|$, has a fat crown decomposition $(H, C, R)$, where $H \subseteq N(V(J))$, that can be found in linear time, given $J$.*

**Proof.** This follows from the previous Lemma. If we replace each $K_2$ with a single vertex, then by Lemma 9.4.3 this graph admits a crown-decomposition. We can reintroduce the $K_2$'s to obtain a fat crown. □

**Lemma 9.4.4** *Any graph $G$ with an independent set $I$, where $|I| \geq 2|N(I)|$, has a double crown decomposition $(H, C, R)$, where $H \subseteq N(I)$, that can be found in linear time given $I$.*

**Proof.** Let $G$ be a graph with an independent set $I \subseteq V(G)$ such that $2|N(I)| \leq |I|$. Create an identical graph $G'$, but for every vertex $v \in N(I)$ add a copy $v'$, such that $N(v) = N(v')$. By Lemma 9.4.3, $G'$ has a crown-decomposition $(H, C, R)$ such that $H \subseteq N_{G'}(I)$. We now claim that we can 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 $v \in H$ but $v' \notin H$. The vertex $v$ must be matched to some vertex $u$ in $C$. Since $N(v) = N(v')$, we have that $v'$ cannot be in $C$ as it would contradict the fact that $C$ is an independent set. Also $v'$ cannot be in $R$, as that would contradict that $H$ is a cut-set. Thus $v'$ must be in $H$, contradicting the assumption.

With this observation, the result follows easily 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 $u_2$. Do this for every pair in $H$. It is easy to see that this forms a double crown in $G$. □

We will now describe a polynomial time preprocessing algorithm that reduces the graph to a kernel of size at most $15k$. The process below either reduces the graph or produces a packing of the appropriate size, thus we can reach a kernel by repeating the following three steps:

```
Step 1.  Compute an arbitrary maximal P2-packing
         W.  Let Q = V \ W.
Step 2.  Let X be the collection of components
         in G[Q] isomorphic to K2.  If |X| ≥ |N(X)|
         in G then reduce by Lemma 9.4.2.
Step 3.  Let I be the isolated vertices in G[Q].
         If |I|  ≥  2|N(I)| in G, then reduce by
         Lemma 9.4.1.
```

**Lemma 9.4.5** *If $|V(G)| > 15k$ then the preprocessing algorithm will either find a $k$-$P_2$-packing or it will reduce $G$.*

**Proof.** Assume in contradiction to the stated lemma that $|V(G)| > 15k$, but that the algorithm produces neither a $k$-$P_2$-packing nor a reduction of $G$.

By the assumption the maximal packing $W$ is of size $|W| < 3k$. Let $Q = V \setminus W$. Let $Q_i$ be the vertices in $Q$ that have degree $i$ in the graph induced by $Q$.

**Claim 5** $\forall i \geq 2, Q_i = \emptyset$

*Proof of Claim 5.* This is clear as otherwise $W$ could not be maximal. □

**Claim 6** $|Q_1| \leq 6k$

*Proof of Claim 6.* Assume in contradiction that $|Q_1| > 6k$. This implies that the number of $K_2$s $X$ in $Q$ is greater than $3k$, but then $|X| > |W|$. By Corollary 9.4.1 $G$ has a 'fat crown' and should have been reduced in step 2 of the algorithm, contradicting that no reduction took place. □

**Claim 7** $|Q_0| \leq 6k$

*Proof of Claim 7.* Assume in contradiction that $|Q_0| > 6k$, but then $|Q_0|$ is more than $2|W|$ and by Lemma 9.4.4 $G$ has a 'double crown' and by Lemma 9.4.1 should have been reduced in step 3 of the algorithm, contradicting that no reduction took place. □

Thus the total size $|V(G)|$ is $|W| + |Q_0| + |Q_1| + |Q_2| + \cdots \leq 3k + 6k + 6k + 0 = 15k$. This contradicts the assumption that $|V(G)| > 15k$. □

**Corollary 9.4.2** *Any instance $(G, k)$ of $P_2$-packing can be reduced to a problem kernel of size $\mathcal{O}(k)$.*

**Proof.** This follows from the Lemma, as we can run the preprocessing algorithm until it fails to reduce $G$. By Lemma 9.4.5, the size is then at most $15k$. □

## 9.5 RUNNING TIME

For computing the kernel, we will run the preprocessing algorithm $\mathcal{O}(n)$ times. Since a maximal $k$-packing of $P_2$'s can be computed in $\mathcal{O}(kn)$ time, the most time consuming part is the $\mathcal{O}(|V| + |E|)$ time needed to compute a crown decomposition. Thus the kernelization process can be completed in $\mathcal{O}(n^3)$ time.

We will apply a straightforward brute-force algorithm on the kernels to find the optimal solution. In the case of $P_2$-packing, we will select the center-vertices of the $P_2$'s in a brute force manner. There are $\binom{15k}{k}$ ways to do this. By Stirling's formula this expression 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 vertices on 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 now have $15k + k$ vertices, and thus $\mathcal{O}(k^2)$ edges. We can solve each of these in time $\mathcal{O}(k^{2.5})$, giving a running time of $\mathcal{O}(2^{5.301k}k^{2.5})$ for the kernel. In total we can decide the $P_2$-packing problem in time $\mathcal{O}(2^{5.301k}k^{2.5} + n^3)$.

Applying the same technique for the $s$-stars we will achieve $\mathcal{O}(2^{\mathcal{O}(k \log k)}k^{\mathcal{O}(1)}n^{\mathcal{O}(1)})$, which is asymptotically worse due to the quadratic kernel.

## 9.6 CONCLUSIONS AND FURTHER RESEARCH

Packing vertex-disjoint copies of a graph $H$ into another graph $G$ is NP-complete as long as $H$ has more than two vertices [HK78]. We have analyzed within the framework of parameterized complexity a specific instance of this problem, the packing of vertex-disjoint stars with $s$ leaves. We have proved that packing $K_{1,2}$'s in a graph $G$, and equivalently $k$-$P_2$-PACKING, has a linear kernel.

Our algorithm for $k$-$P_2$-PACKING runs in time $\mathcal{O}(2^{5.301k}k^{2.5} + n^3)$. This running time arises from reducing the problem to a kernel of size $15k$. We believe that this kernel can be further improved and thus the running time substantially decreased. However, it is already much better than $2^{\mathcal{O}(|H|k \log k + k|H| \log |H|)}$, the running time of the general algorithm in [FHRST04].

We have also proved that $s$-Star Packing ($K_{1,s}$-Packing) is in general fixed-parameter tractable with a quadratic kernel size. We also gave an algorithm for the general case with running time $\mathcal{O}^*(2^{\mathcal{O}(k \log k)})$, but this is not an improvement over [FHRST04] or [AYZ95].

There are several related problems that could be considered in light of the techniques used in Section 9.3. The most obvious one is the following:

$k$-$K_{1,s}$-PACKING
INSTANCE: Graph $G = (V, E)$
PARAMETER: $k$
QUESTION: Are there $k$ edge-disjoint instances of $K_{1,s}$ in $G$?

This problem is fixed-parameter tractable when $s$ is 2 or 3 using Robertson and Seymour's Graph Minor Theorem [RS99]: It can be easily proved that its NO-instances are

closed under minors. The issue here is that this method is non-constructive and carries a fast growing function $f(k)$. Possibly, applying similar arguments as those in Section 9.4 would lead to a much better running time.

**Acknowledgements.** We would like to thank Mike Fellows for all the inspiring conversations leading to the completion of this paper.

# BIBLIOGRAPHY


[ACFL04] F. Abu-Khzam, R. Collins, M. Fellows and M. Langston. Kernelization Algorithms for the Vertex Cover Problem: Theory and Experiments. *Proceedings ALENEX 2004*, Springer-Verlag, *Lecture Notes in Computer Science* (2004), to appear.

[AC99] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, M. Protasi. Complexity and Approximation *Springer Verlag* (1999).

[AYZ95] N. Alon, R. Yuster, U. Zwick. Color-Coding, *Journal of the ACM*, Volume 42(4), pages 844–856 (1995).

[BM02] R. Bar-Yehuda, M. Halldórsson, J. Naor, H. Shachnai, I. Shapira. Scheduling Split Intervals. *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 732-741 (2002).

[BK01] 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*, 2001

[CFJ03] B. Chor, M. Fellows, D. Juedes. An Efficient FPT Algorithm for Saving $k$ colors. *Manuscript* (2003).

[DFS99] R. Downey, M. Fellows, U. Stege. Parameterized Complexity: A Framework for Systematically Confronting Computational Intractability. *AMS-DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, Volume 49, pages 49-99 (1999).

[F03] M. Fellows. Blow-Ups, Win/Win's, and Crown Rules: Some new Directions in FPT. *Proceedings 29th Workshop on Graph Theoretic Concepts in Computer Science*, LNCS 2880(2003), pages 1-12.

[FHRST04] M.Fellows, P.Heggernes, F.Rosamond, C. Sloper, J.A.Telle, Exact algorithms for finding k disjoint triangles in an arbitrary graph, *Proceedings 30th Workshop on Graph Theoretic Concepts in Computer Science* (2004) LNCS 3353, pages 235-244

[FM+00] M.R. 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, *Foundations of Software Technology and Theoretical Computer Science*, LNCS 1974 (2000), page 240.

[HK73] J. Hopcroft and R. Karp. An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs. *SIAM Journal on Computing*, 2 pages 225–231 (1973).

[HK78] P. Hell and D. Kirkpatrick. On the complexity of a generalized matching problem. *Proceedings of 10th ACM Symposium on theory of computing*, pages 309–318 (1978).

[HK86] P. Hell and D. Kirkpatrick. Packings by complete bipartite graphs. *SIAM Journal of Algebraic Discrete Methods*, number 7, pages 199–209 (1986).

[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 packing problems. *SIAM Journal of Discrete Mathematics*, number 2, pages 68–72 (1989).

[K94] V. Kann. Maximum bounded H-matching is MAX-SNP-complete. *Information Processing Letters*, number 49, pages 309–318 (1994).

[PS04] E.Prieto, C. Sloper. Creating Crown Structure — The case of Max Internal Spanning Tree. *Submitted*

[RS99] N. Robertson, PD. Seymour. *Graph Minors XX. Wagner's conjecture*, to appear.

# 10

# PACKING TRIANGLES

**Finding $k$ disjoint triangles in an arbitrary graph**[1]

Mike Fellows Pinar Heggernes Frances Rosamond
Christian Sloper Jan Arne Telle

**Abstract**

We consider the *NP*-complete problem of deciding whether an input graph on $n$ vertices has $k$ vertex-disjoint copies of a fixed graph $H$. For $H = K_3$ (the triangle) we give an $\mathcal{O}(2^{2k \log k + 1.869k} n^2)$ algorithm, and for general $H$ an $\mathcal{O}(2^{k|H| \log k + 2k|H| \log |H|} n^{|H|})$ algorithm. We introduce a preprocessing (kernelization) technique based on crown decompositions of an auxiliary graph. For $H = K_3$ this leads to a preprocessing algorithm that reduces an arbitrary input graph of the problem to a graph on $\mathcal{O}(k^3)$ vertices in polynomial time.

## 10.1 INTRODUCTION

For a fixed graph $H$ and an input graph $G$, the $H$-packing problem asks for the maximum number of vertex-disjoint copies of $H$ in $G$. The $K_2$-packing (edge packing) problem, which is equivalent to maximum matching, played a central role in the history of classical computational complexity. The first step towards the dichotomy of 'good' (polynomial-time) versus 'presumably-not-good' (NP-hard) was made in a paper on maximum matching from 1965 [E65], which gave a polynomial time algorithm for that problem. On the other hand, the $K_3$-packing (triangle packing) problem, which is our main concern in this paper, is *NP*-hard [HK78].

Recently, there has been a growing interest in the area of exact exponential-time algorithms for *NP*-hard problems. When measuring time in the classical way, simply by the

[1]This paper appeared at the conference '30th International Workshop of Graph-Theoretic Concepts in Computer Science' and was published in the proceedings [FHRST04].

size of the input instance, the area of exact algorithms for *NP*-hard problems lacks the classical dichotomy of good ($P$) versus presumably-not-good (*NP*-hard) [W03]. However, if in the area of exact algorithms for *NP*-hard problems we instead measure time in the parameterized way, then we retain the classical dichotomy of good (*FPT* - Fixed Parameter Tractable) versus presumably-not-good ($W[1]$-hard) [DF99]. It therefore seems that the parameterized viewpoint gives a richer complexity framework. In fact, a formal argument for this follows from the realization that the non-parameterized viewpoint, measuring time by input size, is simply a special case of the parameterized viewpoint with the parameter chosen to be the input size. Parameterized thus, any problem is trivially *FPT* and the race for the best *FPT* algorithm is precisely the same as the race for the best non-parameterized exact algorithm. Note that for any optimization or decision problem, there are many interesting possibilities for choice of parameter, that can be guided by both practical and theoretical considerations, see for example [F03] for a discussion of five different parameterizations of a single problem. In our opinion, the relevant discussion for the field of exact algorithms for *NP*-hard problems is therefore not "parameterized or non-parameterized?" but rather "which parameter?"

In this paper our focus is on parameterized algorithms for deciding whether a graph $G$ has $k$ disjoint copies of $K_3$, with the integer $k$ being our parameter. On input $(G, k)$, where $G$ is a graph on $n$ vertices, an *FPT* algorithm is an algorithm with runtime $\mathcal{O}(n^{\alpha} f(k))$, for a constant $\alpha$ and an unrestricted function $f(k)$. We want, of course, both $\alpha$ and the growth rate of $f(k)$ to be as small as possible.

A practical spinoff from the field of parameterized algorithms for *NP*-hard problems has been a theoretical focus on the algorithmic technique of preprocessing, well-known from the heuristic algorithms community. In fact, the parameterized problems having *FPT* algorithms are *precisely* the parameterized problems where preprocessing can in polynomial time reduce a problem instance $(G, k)$ to a kernel, *i.e.*, a decision-equivalent problem instance $(G', k')$ where the size of $G'$ is bounded by a function of $k$ (only), and where also $k' \leq k$ [DFS97]. One direction of this fact is trivial, since any subsequent brute-force algorithm on $(G', k')$ would give an overall *FPT* algorithm. In the other direction, assume we have an *FPT* algorithm with runtime $\mathcal{O}(n^{\alpha} f(k))$ and consider an input $(G, k)$ on $n$ vertices. If $n \geq f(k)$ then the runtime of the *FPT* algorithm on this instance is in fact polynomial and can be seen as a reduction to the trivial case. On the other hand, if $n \leq f(k)$ then the instance $(G, k)$ already satisfies the kernel requirements. Note that in this case the kernel size $f(k)$ is exponential in $k$, and a smaller kernel is usually achievable. For this reason, in the field of parameterized algorithms for *NP*-hard problems, it can be argued that there are two distinct races [F03]:

- Find the fastest *FPT* algorithm for the problem.
- Find the smallest polynomial-time computable kernelization for the problem.

In this paper, we enter the parameterized $K_3$-packing problem into both these races, giv-

ing on the one hand an $\mathcal{O}(2^{2k\log k+1.869k}n^2)$ *FPT* algorithm, and on the other hand an $\mathcal{O}(k^3)$ kernelization. Our *FPT* algorithm is derived by an application of a fairly new technique known as greedy localization [JZC04], and our kernelization algorithm by a non-standard application of the very recently introduced notion of Crown Reduction Rules [CFJ03, CFJ04, F03]. We end the paper by asking how well these two results on $K_3$-packing generalize to $H$-packing. It turns out that the *FPT* algorithm generalizes quite easily, giving *FPT* algorithms for deciding whether an input graph $G$ has $k$ disjoint copies of an arbitrary connected $H$. However, we presently do not see how to generalize the kernelization algorithm.

Just in time for the final version of this paper we realized that Theorem 6.3 in [AYZ95] can be used to give a $2^{\mathcal{O}(k)}$ algorithm for graph packing using color coding. However, we still believe our result to be of practical interest as the constants in color coding can be impractical.

The next section gives some basic graph terminology. We then proceed in Sections 3, 4 and 5 with the kernelization results, before continuing with the *FPT* algorithm in Section 6 for $K_3$ and in Section 7 for general $H$.

## 10.2 PRELIMINARIES

We assume simple, undirected, connected graphs $G = (V, E)$, where $|V| = n$. The neighbors of a vertex $v$ are denoted by $N(v)$. For a set of vertices $A \subseteq V$, $N(A) = \{v \notin A \mid uv \in E \text{ and } u \in A\}$, and the subgraph of $G$ induced by $A$ is denoted by $G(A)$. For ease of notation, we will use informal expressions like $G \setminus u$ to denote $G(V \setminus \{u\}, E)$, $G \setminus U$ to denote $G(V \setminus U, E)$, and $G \setminus e$ to denote $(V, E \setminus \{e\})$, where $u$ is a vertex, $U$ is a vertex set, and $e$ is an edge in $G$. A subset $S$ of $V$ is a *separator* if $G \setminus S$ is disconnected.

An *$H$-packing* $W$ of $G$ is a collection of disjoint copies of the graph $H$ in $G$. We will use $V(W)$ to denote the vertices of $G$ that appear in $W$, and $E(W)$ to denote the edges. A *matching* is a $K_2$-packing.

We will in the following two sections describe a set of reduction rules. If any of these rules can be applied to $G$, we say that $G$ is *reducible*, otherwise *irreducible*.

## 10.3 REDUCTION RULES FOR $K_3$-PACKING

Let us start with a formal definition of the problem we are solving:

$k$-$K_3$-PACKING (TRIANGLE PACKING)
INSTANCE: Graph $G = (V, E)$
PARAMETER: $k$
QUESTION: Does $G$ have $k$ disjoint copies of $K_3$?

We say that a graph $G$ has a $k$-$K_3$-packing if the answer to the above question is "yes." In this section, we identify vertices and edges of the input graph that can be removed without affecting the solution of the $k$-$K_3$-PACKING problem.

**Definition 10.3.1** *If vertices $a, b,$ and $c$ induce a $K_3$, we say that vertex $a$* sponsors *edge $bc$. Likewise, edge $bc$* sponsors *vertex $a$.*

We start with two simple observations that also give preprocessing rules useful to delete vertices and edges that cannot participate in any triangle.

**Reduction Rule 4** *If $e \in E$ has no sponsor then $G$ has a $k$-$K_3$-packing $\iff$ $G \setminus e$ has a $k$-$K_3$-packing.*

**Reduction Rule 5** *If $u \in V$ has no sponsor then $G$ has a $k$-$K_3$-packing $\iff$ $G \setminus u$ has a $k$-$K_3$-packing.*

Both observations are trivially true, and let us remove vertices and edges from the graph so that we are left with a graph containing only vertices and edges that could potentially form a $K_3$.

**Reduction Rule 6** *If $u \in V$ sponsors at least $3k-2$ disjoint edges then $G$ has a $k$-$K_3$-packing $\Leftrightarrow$ $G \setminus u$ has a $(k-1)$-$K_3$-packing.*

**Proof.** $(\Rightarrow:)$ This direction is clear as removing one vertex can decrease the number of $K_3$s by at most one.
$(\Leftarrow:)$ If $G \setminus u$ has a $(k-1)$-$K_3$-packing $S$, then $S$ can use vertices from at most $3(k-1) = 3k-3$ of the disjoint edges sponsored by $u$. This leaves at least one edge that can form a $K_3$ with $u$, thus raising the number of $K_3$s to $k$. □

## 10.4 REDUCING INDEPENDENT SETS - CROWN REDUCTION

In this section we will first give a trivial reduction rule that removes a specific type of independent sets. This reduction rule is then generalized and replaced by a more powerful rule that allows us to reduce any large independent set in the graph.

**Reduction Rule 7** *If $\exists u, v \in V$ such that $N(u) = N(v) = \{a, b\}$ and $ab \in E$, then $G$ has a $k$-$K_3$-packing $\Leftrightarrow$ $G \setminus u$ has a $k$-$K_3$-packing.*

**Proof.** This is trivial as it is impossible to use both $u$ and $v$ in any $K_3$-packing. □

This reduction rule identifies a redundant vertex and removes it. The vertex is redundant because it has a stand-in that can form a $K_3$ in its place and there is no use for both vertices. Generalizing, we try to find a set of vertices such that there is always a distinct stand-in for each vertex in the set.

**Definition 10.4.1** *A* crown decomposition $(H, C, R)$ *in a graph* $G = (V, E)$ *is a partitioning of the vertices of the graph into three sets* $H$*,* $C$*, and* $R$ *that have 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. $C = C_u \cup C_m$ (the crown) *is an independent set in* $G$*.*
3. $|C_m| = |H|$*, and there is a perfect matching between* $C_m$ *and* $H$*.*

Crown-decomposition is a recently introduced idea that supports nontrivial and powerful preprocessing (reduction) rules for a wide variety of problems, and that performs very well in practical implementations [CFJ03, F03, ACFL04]. It has recently been shown that if a graph admits a crown decomposition, then a crown decomposition can be computed in polynomial time [AS04]. The following theorem can be deduced from [CFJ03, page 7], and [F03, page 8].

**Theorem 10.4.1** *Any graph* $G$ *with an independent set* $I$*, where* $|I| \geq n/2$*, has a crown decomposition* $(H, C, R)$*, where* $H \subseteq N(I)$*, that can be found in linear time, given* $I$*.*

For most problems, including $k$-$K_3$-PACKING, it is not clear how a crown decomposition can directly provide useful information. We introduce here the idea of creating an auxiliary graph model where a crown decomposition in the auxiliary graph is used to identify preprocessing reductions for the original graph.

For $k$-$K_3$-PACKING we will show that an auxiliary graph model can be created to reduce large independent sets in the problem instance. Consider an independent set $I$ in a graph $G$. Let $E_I$ be the set of edges that are sponsored by the vertices of $I$.

The auxiliary model that we consider is a bipartite graph $G_I$ where we have one vertex $u_i$ for every vertex $v_i$ in $I$ and one vertex $f_j$ for every edge $e_j$ in $E_I$. For simplicity, we let both sets $\{e_j \mid e_j \in E_I\}$ and $\{f_j \mid e_j \in E_I\}$ be denoted by $E_I$. The edges of $G_I$ are defined as follows: let $u_i f_j$ be an edge in $G_I$ if and only if $u_i$ sponsors $f_j$.

We now prove the following generalization of Reduction Rule 7. This rule now replaces rule 7.

**Reduction Rule 8** *If $G_I$ has a crown decomposition $(H, C_m \cup C_u, R)$ where $H \subseteq E_I$ then $G$ has a $k$-$K_3$-packing $\Leftrightarrow$ $G \setminus C_u$ has a $k$-$K_3$-packing.*

**Proof.** Assume on the contrary that $G_I$ has a crown decomposition $(H, C_m \cup C_u, R)$, where $H \subseteq E_I$ and $G$ has a $k$-$K_3$-packing $W^*$ but $G \setminus C_u$ has no $k$-$K_3$-packing. This implies that some of the vertices of $C_u$ were used in the $k$-$K_3$-packing $W^*$ of $G$.

Let $H^*$ be the set of vertices in $H$ whose corresponding edges in $G$ use vertices from $C = C_m \cup C_u$ to form $K_3$s in the $k$-$K_3$-packing $W^*$ of $G$. Note that vertices in $C_u$ can only form $K_3$s with edges of $G$ that correspond to vertices in $H$. Observe that each edge corresponding to a vertex in $H^*$ uses exactly one vertex from $C$. Further, $|H^*| \leq |H|$. By these two observations it is clear that every edge whose corresponding vertex is in $H^*$ can be assigned a vertex from $C_m$ to form a $K_3$. Thus $C_u$ is superfluous, contradicting the assumption. □

**Observation 10.4.1** *If a bipartite graph $G = (V \cup V', E)$ has two crown decompositions $(H, C, R)$ and $(H', C', R')$ where $H \subseteq V$ and $H' \subseteq V$, then $G$ has a crown decomposition $(H'' = H \cup H', C'' = C \cup C', R'' = R \cap R')$.*

It is easy to check that all properties of a crown decomposition hold for $(H'', C'', R'')$.

**Lemma 10.4.1** *If $G$ has an independent set $I$ such that $|I| > 2|E_I|$ then we can in polynomial time find a crown decomposition $(H, C_m \cup C_u, R)$ where $H \subseteq E_I$, and $C_u \neq \emptyset$.*

**Proof.** Assume on the contrary that $G$ has an independent set $I$ such that $|I| > 2|E_I|$ but $G$ has no crown decomposition with the properties stated in the lemma.

By Theorem 10.4.1 the bipartite model $G_I$ as described above has a crown decomposition $(H, C = C_m \cup C_u, R)$ where $H \subseteq N(I)$ and consequently $C \subseteq I$. If $|I \setminus C| > |E_I|$ then $G_I \setminus C$ has a crown decomposition $(H', C', R')$, where $H' \subset N(I)$. By Observation 10.4.1 $(H, C, R)$ and $(H', C', R')$ could be combined to form a bigger crown. Let $(H'', C'' = C''_m \cup C''_u, R'')$ be the largest crown decomposition that can be obtained by repeatedly finding a new crown in $I \setminus C$ and combining it with the existing crown decomposition to form a new head and crown.

By our assumption $C''_u = \emptyset$. Since $|C''_m| = |H''| \leq E_I$ and it follows from Theorem 10.4.1 that $|I \setminus C''_m| \leq |E_I|$ (otherwise a new crown could be formed), we have that $|I| = |C''_m| + |I \setminus C''_m| \leq |E_I| + |E_I| \leq 2|E_I|$, contradicting the assumption that $|I| > 2|E_I|$. □

## 10.5 COMPUTING A CUBIC KERNEL

We now introduce a polynomial time algorithm that either produces a $k$-$K_3$-packing or finds a valid reduction of any input graph $G = (V, E)$ of at least a certain size. We show that this algorithm gives an $\mathcal{O}(k^3)$ kernel for $k$-$K_3$-PACKING.

The algorithm has the following steps:

1. Reduce by Rule 1 and 2 until neither apply.
2. Greedily, find a maximal $K_3$-packing W in $G$. If $|V(W)| \geq 3k$ then ACCEPT.
3. Find a maximal matching $Q$ in $G \setminus V(W)$. If a vertex $v \in V(W)$ sponsors more than $3k - 3$ matched edges, then $v$ can be reduced by Reduction Rule 6.
4. If possible, reduce the independent set $I = V \setminus (V(W) \cup V(Q))$ with Reduction Rule 8.

We now give the following lemma to prove our result:

**Lemma 10.5.1** *If $|V| > 108k^3 - 72k^2 - 18k$, then the preprocessing algorithm will either find a $k$-$K_3$-packing or it will reduce $G = (V, E)$.*

**Proof.** Assume on the contrary to the stated lemma that $|V| > 108k^3 - 72k^2 - 18k$, but that the algorithm produced neither a $k$-$K_3$-packing nor a reduction of $G$.

By the assumption the maximal packing $W$ is of size $|V(W)| < 3k$.

Let $Q$ be the maximal matching obtained by step 2 of the algorithm.

**Claim 1** $|V(Q)| \leq 18k^2 - 18k$

> *Proof of Claim 1.* Assume on the contrary that $|V(Q)| > 18k^2 - 18k$. Observe that no edge in $G \setminus V(W)$ can sponsor a vertex in $G \setminus V(W)$ as this would contradict that $W$ is maximal, therefore all edges in the the maximal matching $Q$ are sponsored by at least one vertex in $V(W)$. If $|V(Q)| > 18k^2 - 18k$, $Q$ contains more than $9k^2 - 9k$ edges. Thus at least one vertex $v \in V(W)$ sponsors more than $(9k^2 - 9k)/3k = 3k - 3$ edges. Consequently $v$ should have been removed by Reduction Rule 6, contradicting the assumption that no reduction of $G$ took place. We have reached a contradiction, thus the assumption that $|V(Q)| > 18k^2 - 18k$ must be wrong. □

Let $I = V \setminus (V(W) \cup V(Q))$. Note that $I$ is an independent set.

**Claim 2** $|I| \leq 108k^3 - 90k^2$

> *Proof of Claim 2.* Assume on the contrary that $|I| > 108k^3 - 90k^2$. Observe that each edge that is sponsored by a vertex of $I$ is either in the subgraph of $G$ induced by $V(W)$, or is an edge between $V(W)$ and $V(Q)$. There are at most $|E_I| = |V(Q)| \cdot |V(W)| + |V(W)|^2 \leq (18k^2 - 18^k) \cdot 3k + (3k)^2 \leq 54k^3 - 45k^2$ such edges.
>
> By Lemma 10.4.1 there are no more than $2|E_I| = 108k^3 - 90k^2$ vertices in $I$, which contradicts the assumption that $|I| > 108k^3 - 90k^2$. □

Thus the total size $|V|$ is $|V(W)| + |V(Q)| + |I| \leq 3k + 18k^2 - 18k + 108k^3 - 90k^2 = 108k^3 - 72k^2 - 18k$. This contradicts the assumption that $|V| > 108k^3 - 72k^2 - 18k$.

□

**Corollary 10.5.1** *Any instance $(G, k)$ of $k$-$K_3$-*PACKING *can be reduced to a problem kernel of size $\mathcal{O}(k^3)$.*

**Proof.** This follows from Lemma 10.5.1, as we can repeatedly run the algorithm until it fails to reduce the graph further. By Lemma 10.5.1 the resulting graph is then of size $\mathcal{O}(k^3)$. □

Note that a $\mathcal{O}(k^3)$ kernel gives us a trivial *FPT*-algorithm by testing all $\mathcal{O}(\binom{k^3}{3k})$ subsets in a brute force manner. This leads to an $\mathcal{O}(2^{9k \log k} + poly(n, k))$ algorithm. However, we will show in the next section that another *FPT* technique yields a faster algorithm.

## 10.6 WINNING THE FPT RUNTIME RACE

In this section we give a faster FPT-algorithm using the technique of Greedy Localization and a bounded search tree.

We begin with the following crucial observation.

**Observation 10.6.1** *Let $W$ be a maximal $K_3$-packing, and let $W^*$ be a $k$-$K_3$-packing. Then for each $K_3$ $T$ of $W^*$, we have $V(T) \cap V(W) \neq \emptyset$.*

**Proof.** Assume on the contrary that there exists a $K_3$ $T$ in $W^*$ such that $V(T) \cap V(W) = \emptyset$. This implies that $V(T) \cup V(W)$ is a $K_3$-packing contradicting that $W$ is a maximal packing. □

**Theorem 10.6.1** *It is possible to determine whether a graph $G = (V, E)$ has a $k$-$K_3$-packing in time $\mathcal{O}(2^{2k \log k + 1.869k} n^2)$.*

**Proof.** Let $W$ be a maximal $K_3$-packing. If $|V(W)| \geq 3k$ we have a $K_3$-packing. Otherwise, create a search tree $T$. At each node we will maintain a collection $S^i = S^i_1, S^i_2, \ldots, S^i_k$ of vertex subsets. These subsets represent the $k$ triangles of the solution, and at the root node all subsets are empty.

From the root node, create a child $i$ for every possible subset $W_i$ of $V(W)$ of size $k$. Let the collection at each node $i$ contain $k$ singleton sets, each containing a vertex of $W_i$.

We say that a collection $S^i = S^i_1, S^i_2, \ldots, S^i_k$ is a *partial solution* of a $k$-$K_3$-packing $W^*$ with $k$ disjoint triangles $W^*_1, W^*_2, \ldots, W^*_k$ if $S^i_j \subseteq V(W^*_j)$ for $1 \leq j \leq k$.

For a child $i$, consider its collection $S_i = S^i_1, S^i_2, \ldots, S^i_k$. Add vertices to $S^i_1$ such that $S^i_1$ induces a $K_3$ in $G$, continue in a greedy fashion to add vertices to $S^i_2, S^i_3$ and so on. If we can complete all $k$ subsets we have a $k$-$K_3$ packing. Otherwise, let $S^i_j$ be the first set which is not possible to complete, and let $V'$ be the vertices we have added to $S^i$ so far. We can now make the following claim.

**Claim 1** *If $S^i = S^i_1, S^i_2, \ldots, S^i_k$ is a partial solution then there exists a vertex $v \in V'$ such that $S^i = S^i_1, \ldots, (S^i_j \cup \{v\}), \ldots, S^i_k$ is a partial solution.*

> *Proof of Claim 1.* Assume on the contrary that $S^i = S^i_1, S^i_2, \ldots, S^i_k$ is a partial solution but that there exists no vertex $v \in V'$ such that $S^i = S^i_1, (S^i_j \cup \{v\}), \ldots, S^i_k$ is a partial solution. This implies that $V(W^*_j) \cap V' = \emptyset$, but then we could add $V(W^*_j) \setminus S^i_j$ to $S^i_j$ to form a new $K_3$, thus contradicting that it was not possible to complete $S^i_j$. □

We now create one child $u$ of node $i$ for every vertex in $u \in V'$. The collection at child $u$ is $S^i = S^i_1, (S^i_j \cup \{u\}), \ldots, S^i_k$. This is repeated at each node $l$, until we are unable to complete any set in node $l$'s collection, i.e., $V' = \emptyset$.

By Observation 10.6.1 we know that if there is $k$-$K_3$-packing then one of the branchings from the root node will have a partial solution. Claim 1 guarantees that this solution is propagated down the tree until finally completed at level $2k$.

At each level the collections $S$ at the nodes grow in size, thus we can have at most $2k$ levels in the search tree. Observe that at height $h$ in the search tree $|V'| < 2k - h$, thus fan-out at height $h$ is limited to $2k - h$. The total size of the tree is then at most $\binom{3k}{k} 2k \cdot (2k-1) \cdot \cdots = \binom{3k}{k} \cdot 2k! = \frac{(3k)!}{k!}$. Using Stirling's approximation and suppressing

some constant factors we have $\frac{(3k)!}{k!} \approx 3.654^k \cdot k^{2k} = 2^{2k \log k + 1.869k}$. At each node we need $\mathcal{O}(n^2)$ time to maximize the sets. Hence, the total running time is $\mathcal{O}(2^{2k \log k + 1.869k} n^2)$ □

Note that it is, of course, possible to run the search tree algorithm from this section on the kernel obtained in the previous section. The total running time is then $\mathcal{O}(2^{2k \log k + 1.869k} k^6 + p(n,k))$. This could be useful if $n$ is much larger than $k$ as the additive exponential (rather than multiplicative) factor becomes significant.

## 10.7 PACKING ARBITRARY GRAPHS

In their paper from 1978, Hell and Kirkpatrick [HK78] prove that $k$-$H$-packing for any connected graph $H$ of 3 or more vertices is *NP*-complete. We will in this section show that our search tree technique for $k$-$K_3$-packing easily generalizes to arbitrary graphs $H$, thus proving that packing any subgraph is in FPT.

$k$-$H$-PACKING
INSTANCE: Graph $G = (V, E)$
PARAMETER: $k$
QUESTION: Does $G$ have at least $k$ disjoint copies of $H$?

**Theorem 10.7.1** *It is possible to determine whether a graph $G = (V, E)$ has a $k$-$H$-packing in time $\mathcal{O}(2^{k|H| \log k + 2k|H| \log |H|} n^{|H|})$.*

**Proof.** The proof is analogous to the proof of Theorem 10.6.1. However, as we no longer can depend upon perfect symmetry in $H$ (since $H$ is not necessarily complete), we must maintain a collection of ordered sequences at each tree-node. Each sequence represents a partial $H$-subgraph.

The possible size of $V'$ increases to $k|H| - k$. Then when we want to determine which $v$ of $V'$ to add to the sequence, we must try every $v$ in every position in $H$. Thus the fan-out at each node increases to $k|H|^2 - k|H|$. The height of the tree likewise increases to at most $k|H| - k$. Thus the new tree size is $\binom{k|H|}{k}(k|H|^2 - k|H|)^{k|H|-k}$, which is strictly smaller than $k^{k|H|}|H|^{2k|H|}$ or $2^{k|H| \log k + 2k|H| \log |H|}$. □

## 10.8 SUMMARY AND OPEN PROBLEMS

Our main results in the two *FPT* races are:

(1) We have shown an $\mathcal{O}(k^3)$ problem kernel for the problem of packing $k$ triangles.

(2) We have shown that for any fixed graph $H$, the problem of packing $k$ $H$s is in *FPT*

with a parameter function of the form $\mathcal{O}(2^{O(k \log k)})$ and more practical constants than [AYZ95].

In addition to "upper bound" improvements to these initial results, which would be the natural course for further research - now that the races are on - it would also be interesting to investigate lower bounds, if possible.

It would be interesting to investigate the "optimality" of the form of our *FPT* results in the sense of [CJ03, DEFPR03]. Can it be shown that there is no $\mathcal{O}(2^{o(k)})$ *FPT* algorithm for $k$-$H$-PACKING unless *FPT*$= M[1]$?

Many parameterized problems admit linear problem kernels. In fact, it appears that most naturally parameterized problems in *APX* are in *FPT* and have linear problem kernels. However, it seems unlikely that *all FPT* problems admit linear kernels. We feel that $k$-$K_t$-PACKING is a natural candidate for an *FPT* problem where it may not be possible to improve on $\mathcal{O}(k^t)$ kernelization. Techniques for the investigation of lower bounds on kernelization are currently lacking, but packing problems may be a good place to start looking for them.

# BIBLIOGRAPHY


[AS04] F. AbuKhzam and H. Suters, Computer Science Department, University of Tennessee, Knoxville, private communications, Dec. 2003.

[AYZ95] N. Alon, R. Yuster, U. Zwick. Color-Coding. J. ACM, pp. 844-856, 1995

[ACFL04] F. Abu-Khzam, R. Collins, M. Fellows and M. Langston. Kernelization Algorithms for the Vertex Cover Problem: Theory and Experiments. *Proceedings ALENEX 2004*, Springer-Verlag, *Lecture Notes in Computer Science* (2004), to appear.

[CFJ03] B. Chor, M. Fellows, and D. Juedes. Saving $k$ Colors in Time $\mathcal{O}(n^{5/2})$. Manuscript, 2003.

[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 of WG2004, Springer-Verlag, *Lecture Notes in Computer Science* (2004)

[CJ03] L. Cai and D. Juedes. On the existence of subexponential parameterized algorithms. *Journal of Computer and System Sciences* 67 (2003).

[DEFPR03] R. Downey, V. Estivill-Castro, M. Fellows, E. Prieto-Rodriguez and F. Rosamond. Cutting Up is Hard to Do: the Parameterized Complexity of $k$-Cut and Related Problems. Electronic Notes in Theoretical Computer Science 78 (2003), 205–218.

[DF99] R. Downey and M. Fellows. *Parameterized Complexity* Springer-Verlag (1999).

[DFS97] R. Downey, M. Fellows and U. Stege, Parameterized Complexity: A Framework for Systematically Confronting Computational Intractability, in: *Contemporary Trends in Discrete Mathematics*, (R. Graham, J. Kratochvil, J. Nesetril and F. Roberts, eds.), AMS-DIMACS Series in Discrete Mathematics and Theoretical Computer Science 49, pages 49-99, 1999.

[E65] J.Edmonds. Paths, trees and flowers, *Can.J.Math.*, 17, 3, pages 449-467, 1965.

[F03] M.Fellows. Blow-ups, Win/Wins and Crown Rules: Some New Directions in *FPT*. *Proceedings WG 2003*, Springer Verlag LNCS 2880, pages 1-12, 2003.

[HK78] P. Hell and D. Kirkpatrick. On the complexity of a generalized matching problem. *Proceedings of 10th ACM Symposium on theory of computing*, pages 309-318, 1978.

[HS89] C. A. J. Hurkens, and A. Schrijver. On the size of systems of sets every t of which have an SDR, with an application to the worst-case ratio of heuristics for packing problems, *SIAM J. Disc. Math. 2*, pages 68-72, 1989.

[JZC04] W. Jia, C. Zhang and J. Chen. An efficient parameterized algorithm for $m$-set packing, *Journal of Algorithms*, 50(1):106–117, 2004.r.

[K91] V. Kann. Maximum bounded 3-dimensional matching is MAX S*NP*-complete, *Inform. Process. Lett*. 37, pages 27-35, 1991.

[W03] G. Woeginger. Exact algorithms for *NP*-hard problems: A survey, *Combinatorial Optimization - Eureka! You shrink!*, M. Juenger, G. Reinelt and G. Rinaldi (eds.). LNCS 2570, Springer, pages 185-207, 2003.

# 11

# FIXED PARAMETER SET SPLITTING

**Fixed Parameter Set Splitting, Linear Kernel and Improved Running Time**[1]

Daniel Lokshtanov  Christian Sloper

**Abstract**

We study the problem k-SET SPLITTING in fixed parameter complexity. We show that the problem can be solved in time $\mathcal{O}^*(2.6494^k)$, improving on the best currently known running time of $\mathcal{O}^*(8^k)$. This is done by showing that a non-trivial instance must have a small minimal SET COVER, and using this to reduce the problem to a series of small instances of MAX SAT.

We also give a linear kernel containing $2k$ elements and $2k$ sets. This is done by reducing the problem to a bipartite graph problem where we use crown decomposition to reduce the graph. We show that this result also gives a good kernel for MAX CUT.

## 11.1 INTRODUCTION

The problem we study in this short note is MAXIMUM SET SPLITTING. The transformation from MAXIMUM SET SPLITTING to MAX CUT preserves the parameter and thus our kernel applies for this problem as well.

$k$-SET SPLITTING
INSTANCE: A tuple $(X, \mathcal{F}, k)$ where $\mathcal{F}$ is a collection of subsets of a finite set $X$, and a positive integer $k$
PARAMETER: $k$

[1]This paper appeared at the conference 'Algorithms and Complexity in Durham', 2005 and has later been invited to a special issue of Journal of Discrete Algorithms [LS05].

> QUESTION: Is there a subfamily $\mathcal{F}' \subseteq \mathcal{F}, |\mathcal{F}'| \geq k$, and a partition of $X$ into disjoint subsets $X_0$ and $X_1$ such that for every $S \in \mathcal{F}'$, we have $S \cap X_0 \neq \emptyset$ and $S \cap X_1 \neq \emptyset$?

SET SPLITTING, or HYPERGRAPH COLORING as it is named in some sources, is a well studied problem. A decision version of the problem appears in [GJ79] as problem [SP4]. It is APX-complete [Pe94] and there have been several approximation algorithms published. The most notable are Anderson and Engebretsen [AE97] with a factor of $0.7240$, and Zhang and Ling [ZL01] with a factor of $0.7499$.

In the area of parameterized algorithms there have been several results published. The first by Dehne, Fellows, and Rosamond [DFR03] who give a $\mathcal{O}^*(72^k)$ FPT algorithm. Dehne, Fellows, Rosamond, and Shaw [DFRS04] then improved on this result giving a $\mathcal{O}^*(8^k)$ algorithm using a combination of the techniques *greedy localization* and *crown decomposition*.

To improve the running time we show that any non-trivial solution of SET SPLITTING has a SET COVER of size at most $k$. We can then reduce the problem to $2^k$ instances of MAX SAT with $k$ clauses each. By using Chen and Kanj's [CK04] exact algorithm with running time $\mathcal{O}^*(1.3247^k)$ on each instance, we get a total running time of $\mathcal{O}^*(2.6494^k)$.

We will also show how we can use crown decomposition to obtain a linear kernel. We do this by reducing the problem to a bipartite graph problem, BIPARTITE COLORFUL NEIGHBORHOOD. We will use crown decomposition to reduce the graph; then show that a simple greedy algorithm decides instances where $k \leq |\mathcal{F}|/2$. Together the two results give a linear kernel with at most $2k$ elements and at most $2k$ sets.

## 11.2 PRELIMINARIES

We assume that in a SET SPLITTING instance every set contains at least two elements of $X$. This is a natural assumption as sets of size one cannot be split in any case.

We employ the $\mathcal{O}^*$ notation introduced in [W03], which suppresses the polynomials in the running time and focus on the exponentials. Thus for a $\mathcal{O}^*(2^k)$ algorithm, there exists a constant $c$ such that the running time is $\mathcal{O}(2^k n^c)$.

Throughout the text we will use lower case letters for elements, edges and vertices, capitals for sets, and caligraphy for sets of sets, i.e., $x, X, \mathcal{X}$, respectively.

In graphs, the set of neighbors of a vertex $v$ is denoted $N(v)$, and the neighbors of a set $S \subseteq V$ is denoted $N(S) = \bigcup_{v \in S} N(v) - S$.

## 11.3 USING SET COVER TO IMPROVE RUNNING TIME

Let a set cover be a subset $S \subseteq X$ such that for every set $P \in \mathcal{F}$, we have $P \cap S \neq \emptyset$. We will prove that an instance either has a set cover of size $k$ or it has a $k$-SET SPLITTING. As we will show, obtaining a small set cover allows us to reduce the problem to a series of MAX SAT problems.

**Lemma 11.3.1** *Any instance $(X, \mathcal{F}, k)$ of Set Splitting that has a* minimal *set cover $S$, has a partitioning of $X$ into disjoint subsets $X_0$ and $X_1$ such that at least $|S|$ sets are split.*

**Proof.** Let $S = \{s_1, s_2, s_3, \ldots, s_n\}$ be a minimal set cover in $(X, \mathcal{F}, k)$. By minimality of $S$, we have that for all $s_i \in S$ there is a set $P_i \in \mathcal{F}$ such that $S \cap P_i = \{s_i\}$. Since every set is of size at least two we can obtain a split of each of these sets $P_i$ by partitioning $X_0 = S$ and $X_1 = X - S$. □

We will now show that we can solve the problem of set splitting by creating at most $2^k$ small instances (at most $k$ clauses) of MAX SAT.

> MAX SAT
> INSTANCE: A collection $\mathcal{C}$ of clauses over a set of variables $X$
> QUESTION: What is the truth assignment that satisfies the maximum number of clauses?

A recent paper by Chen and Kanj [CK04] gives a $\mathcal{O}^*(1.3247^m)$ algorithm for MAX SAT where $m$ is the number of clauses in the formula. We will use this algorithm to solve our MAX SAT instances.

**Theorem 11.3.1** *Set Splitting can be solved in time $\mathcal{O}^*(2.6494^k)$*

**Proof.** We obtain a minimal set cover $S$ by greedily selecting vertices to cover all sets. By Lemma 11.3.1 we know that $S$ has size less than $k$, otherwise we can immediatly answer 'Yes'. Let $\mathcal{P} = \{P \mid P \in \mathcal{F}, P \not\subseteq S\}$. It is clear that $|\mathcal{P}| < k$, otherwise the partition $(S, X \setminus S)$ splits at least $k$ sets. The remaining sets are only affected by how we partition $S$.

Observe that if $S$ was already partitioned into disjoint subsets $X'_0, X'_1$ every set in $\mathcal{P}$ has at least one member in $X'_0$ or in $X'_1$.

Assume we have a partitioning $(X'_0, X'_1)$ of $S$. For each set $R \in \mathcal{P}$, where $R$ is not split by $X'_0$, and $X'_1$, create a clause $C_R$. If $R$ contains an element in $X'_0$ add literals $x_i$ for each

element $x_i \in R - S$ to $C_R$. If $R$ contains an element in $X'_1$, then add literals $\overline{x_i}$, for each element $x_i \in R - S$ to $C_R$.

Adding an element $x$ to $X'_0$ now corresponds to setting variable $x$ false, and vice versa. Observe that a set $R \in \mathcal{P}$ is split if and only if its clause $C_R$ is satisfied. We can now employ Chen and Kanj's exact algorithm for MAX SAT. There are $2^k$ different partitions of the set cover $S$, for each we construct an instance of MAX SAT with at most $k$ clauses. Thus we get a total running time of $\mathcal{O}^*(2^k \cdot 1.3247^k) = \mathcal{O}^*(2.6494^k)$. □

## 11.4 REDUCING TO A GRAPH PROBLEM

The running time of the algorithm in the previous section is multiplicative, i.e., of the form $\mathcal{O}(f(k) \cdot n^c)$. It is often advantagous to have the exponential function as an additive term of the form $\mathcal{O}(f(k) + n^c)$. We can achieve this by reducing, in polynomial time, the problem to a kernel. A *kernel* is a smaller instance of the same problem where the size of the instance is bounded by a function $g(k)$. If $g(k)$ is a linear function we call the kernel a *linear kernel*. Having a linear kernel is often advantagous when designing brute force algorithms for a problem. In this section we show how a linear kernel can be achieved using *crown decompositon*.

Recently the fixed parameter kernels for many problems have been improved using crown decompositions. It is a common technique [FHRST04, PS04] to create an auxiliary graph model from the problem instance and then show that a reduction (using crown decomposition) in the graph model leads to reduction of the problem instance. This technique would apply to this problem, but we will instead reduce our problem to a problem on bipartite graphs.

We reformulate the problem as a problem on bipartite graphs. Let $G(V_\mathcal{F}, V_X, E)$ be a bipartite graph, where $V_\mathcal{F}$ is a set of vertices with a vertex $v_M$ for each set $M \in \mathcal{F}$, and $V_X$ is a set of vertices with a vertex $v_x$ for each element $x \in X$ and let $(v_x, v_M) \in E$ be an edge if $x \in M$.

The problem is now reduced to color the set $V_X$ black and white such that at least $k$ vertices of $V_\mathcal{F}$ have a *colorful neighborhood*, i.e., at least one neighbor of each color. It is easy to see that this problem is equivalent to $k$-SET SPLITTING.

> $k$-BIPARTITE COLORFUL NEIGHBORHOOD ($k$-BCN)
> INSTANCE: A bipartite graph $G = (V_\mathcal{F}, V_X, E)$, and a positive integer $k$
> PARAMETER: $k$
> QUESTION: Is there a two-coloring of $V_X$ such that there exists a set $S \subseteq V_\mathcal{F}$ of size at least $k$ where each element of $S$ has a colorful neighborhood?

As mentioned we will use crown decomposition to reduce the problem. Crown decompo-

sition is particularly well suited for use in bipartite graphs, as Lemma 11.4.1 ensures us the existence of a crown decomposition in any bipartite graph.

**Definition 11.4.1** *A* crown decomposition $(H, C, R)$ *in a graph* $G = (V, E)$ *is a partitioning of the vertices of the graph into three sets* $H$*,* $C$*, and* $R$ *where* $H$ *and* $C$ *are nonempty such that they have the following properties:*

1. $H$ (the head) *is a vertex separator in* $G$*, such that there are no edges in* $G$ *between vertices belonging to* $C$ *and vertices belonging to* $R$*.*
2. $C = C_u \cup C_m$ (the crown) *is an independent set in* $G$*.*
3. *There is a bijective mapping* $f : H \to C_m$*, where* $f(v) = u \Rightarrow (u, v) \in E$ *(i.e., a perfect matching).*

We can find the following lemma in [CFJ04].

**Lemma 11.4.1** *If a graph* $G = (V, E)$ *has an independent set* $I \subseteq V(G)$ *such that* $|N(I)| < |I|$ *then a crown decomposition* $(H, C, R)$ *with* $C \subseteq I$ *for* $G$ *can be found in time* $\mathcal{O}(|V| + |E|)$*.*

Our main reduction rule is the following lemma that states that any crown decomposition can be transformed to a crown decomposition where the head and crown can be removed from the graph.

**Lemma 11.4.2** *Given a bipartite graph* $G = (V_{\mathcal{F}}, V_X, E)$ *where* $|V_{\mathcal{F}}| < |V_X|$*, there exists a nontrivial crown decomposition* $(H, C, R)$ *such that* $G$ *is a 'Yes'-instance for* $k$*-BCN* $\iff$ $G' = (V_{\mathcal{F}} \setminus H, V_X - C, E)$ *is a 'Yes'-instance for* $(k - |H|)$*-BCN*

**Proof.** Since $|V_{\mathcal{F}}| < |V_X|$ there exists a component $V'_{\mathcal{F}} \subseteq V_{\mathcal{F}}, V'_X \subseteq V_X$ where $|V'_{\mathcal{F}}| < |V_X|'$. By Lemma 11.4.1 we know that this component has a crown decomposition $(H', C', R')$ where $H' \subseteq V'_{\mathcal{F}}$. We now use this crown to identify another crown $(H, C, R)$ with the desired properties.

We assume $R \neq \emptyset$, if this is not the case we can move a vertex from $C_u$ to $R$. If $C_u \cup R = \emptyset$ then $|V'_{\mathcal{F}}| = |V'_X|$, contradicting $|V'_{\mathcal{F}}| < |V'_X|$.

We iteratively compute this new crown in the following manner. Let $H_0 \subseteq H'$ be the set of vertices of $H'$ that have a neighbor in $V_X - C$. The set $H_0$ is nonempty since $R \neq \emptyset$ and $H'$ is a vertex separator. Let $C_0$ be the vertices of $C$ that are matched to

$H_0$. Let $H_{i+1} = N(C_i)$ and $C_{i+1}$ be the vertices matched to $H_{i+1}$. Run iteratively until $H_{i+1} = H_i$ then let $H = H_i, C = \{v \mid v \in V_X, N(v) \subseteq H\}$ and $R$ be the remainder.

From the construction of $(H, C, R)$ it is clear that this is a crown decomposition. We proceed to show that $G$ is a Yes-instance for $k$-BCN if and only if $G' = (V_{\mathcal{F}} - H, V_X - C, E)$ is a YES instance for $(k - |H|)$-BCN.

In one direction assume on the contrary that $G$ is a Yes- instance for $k$-BCN, but that $G' = (V_{\mathcal{F}} - H, V_X - C, E)$ is a No instance for $(k - |H|)$-BCN. Then the removed elements $C$ must have participated in a colorful neighborhood for more than $|H|$ vertices in $V_{\mathcal{F}}$. This is clearly impossible as $N(C) \subseteq H$.

In the other direction we have that $G' = (V_{\mathcal{F}} - H, V_X - C, E)$ is a Yes-instance for $(k - |H|)$-BCN. We can assume that every vertex in $V_X - C$ has been colored. We can now color $C$ such that every vertex in $H$ has a colorful neighborhood. For every vertex $h \in H_0$ we can color the vertex matched to $h$ different from $h$'s neighbor in $V_X - C$. Observe that after coloring $C_j$, all vertices in $H_{j+1} - H_j$ have a neighbor in $C_j$. Thus we can obtain a colorful neighborhood for each vertex $h \subseteq H_{j+1} - H_j$ by coloring its matched vertex appropriately. Thus every vertex in $H$ has a colorful neighborhood and $G$ is a YES instance for $k$-BCN. □

We say that a bipartite graph is *irreducible* if we cannot apply the reduction in Lemma 11.4.2. The following corollary follows directly.

**Corollary 11.4.1** *In an irreducible bipartite graph $G = (|V_F|, |V_X|, E)$, we always have $|V_X| \leq |V_{\mathcal{F}}|$.*

We have obtained the inequality $|V_X| \leq |V_{\mathcal{F}}|$. We now show that we can obtain a similar relationship between $|V_{\mathcal{F}}|$ and $k$ by analyzing the effectiveness of a simple greedy algorithm for the problem.

Greedy algorithms for SET SPLITTING seem to do quite well, and it is indeed possible to prove that there is a polynomial time algorithm that splits at least half of the sets. For our graph problem this is the equivalent of proving that it is always possible to two-color $V_X$ such that at least half of $V_{\mathcal{F}}$ has a colorful neighborhood.

**Lemma 11.4.3** *It is always possible to find a partitioning $(B, W)$ of $V_X$ such that at least half of the vertices in $V_{\mathcal{F}}$ have a colorful neighborhood.*

**Proof.** For a subset $V'_X \subseteq V_X$ we define $M(V'_X) = \{v_M \mid v_M \in V_{\mathcal{F}}, N(v_M) \subseteq V'_X\}$. We proceed by induction on the size of $V'_X$.

**Base case:** If $|V'_X| = 1$, then $M(V'_X) = \emptyset$. Thus the statement is trivially true.

**Inductive Hypothesis:** We assume that for all sets $V'_X \subseteq V_X$ of size $n_0$ we can find a partitioning $B', W'$ of $V'_X$ such that at least half of the vertices in $M(V'_X)$ has a colorful neighborhood.

**Inductive Step:** Assume any set $V''_X \subseteq V_X$ where $|V''_X| = n_0 + 1$. Let $v_x \in V''_X$ be an arbitrary vertex in $V''_X$, and let $M' = M(V''_X - v_x)$ . By the inductive hypothesis we can find a partitioning $B', W'$ such that half of the vertices in $M'$ have a colorful neighborhood. Since every vertex in $V_{\mathcal{F}}$ has degree at least 2, every vertex in $M(V''_X) - M'$ has at least one neighbor in $B' \cup W'$. We can assume without loss of generality that half of the vertices of $M(X'') - M'$ have a neighbor in $B'$. Hence the partitioning $B', W' \cup \{v_x\}$ ensures that at least half of the vertices in $M(V''_X)$ have a colorful neighborhood.

□

The following corollary follows directly from the above lemma. It is easy to design a greedy algorithm that mimic the inductive procedure in the proof and produces the necessary partitioning.

**Corollary 11.4.2** *All instances where $k \leq |V_{\mathcal{F}}|/2$ are trivially 'Yes'-instances.*

**Theorem 11.4.1** *$k$-BCN has a linear kernel where $|V_X| \leq |V_{\mathcal{F}}| < 2k$.*

**Proof.** By Corollary 11.4.2 we have that for a nontrivial instance $(G, k), k > |V_{\mathcal{F}}|/2$. By Corollary 11.4.1 we have that $|V_X| \leq |V_{\mathcal{F}}|$ after reducing the graph. Thus the inequality $|V_X| \leq |V_{\mathcal{F}}| < 2k$ holds for the kernel. □

The following corollary then follows by a transformation of the kernel back to $k$-SET SPLITTING.

**Corollary 11.4.3** *$k$-SET SPLITTING has a linear kernel of $2k$ sets and $2k$ elements.*

## 11.5 AN APPLICATION TO MAX CUT

In this section we mention that our kernelization result also applies to the more known MAX CUT, which can be encoded using SET SPLITTING.

MAX CUT
INSTANCE: A graph $G = (V, E)$, and a positive integer $k$
PARAMETER: $k$
QUESTION: Is there a partitioning of $V$ into two sets $V'$, $V''$ such that the number of edges between $V'$ and $V''$ is at least k?

Let the set of elements $X = V$ and for every edge $(v, u) \in E$ create a set $\{v, u\}$. A splitting of a set $vu$ now corresponds to placing $u$ and $v$ in different partitions in MAX CUT. The results on SET SPLITTING thus apply to MAX CUT.

**Observation 11.5.1** *$k$-MAX CUT has a linear kernel of $2k$ vertices and $2k$ edges.*

Using the best known exact algorithm for this problem, an $\mathcal{O}^*(2^{|E|/4})$ algorithm by Fedin and Kulikov [FK02], we get a running time of $\mathcal{O}^*(2^{k/2})$ which is equivalent to Prieto's algorithm in [P04] where she used the *Method of Extremal Structure*, another well known FPT technique, to reach a kernel of $k$ vertices and $2k$ edges. Earlier Mahajan, Raman [MR99] has used yet another technique to reach the same number of edges.

## 11.6 CONCLUSION

We have improved the current best algorithm for SET SPLITTING of $\mathcal{O}^*(8^k)$ to $\mathcal{O}^*(2.6494^k)$ using an observation about the size and structure of the minimal set covers in any set splitting instance.

We also obtained a linear kernel by using modelled crown decomposition. Our model is different from the one seen in [DFRS04]. This shows how crown decompositions can often be applied in many ways to a single problem, with varying results. This kernel also applies to Max Cut equalling the best known kernels for this problem, but with a different approach.

Having achieved a linear kernel for Set Splitting we believe that it is now possible to improve the running time even further. Applying a variation of the transformation seen in the proof of Theorem 11.3.1 it is possible to transform an instance of SET SPLITTING to an instance of Max Sat. Add two clauses for each set, with one literal for each variable. In one clause all literals are positive and in the other all negative. The set is now split if and only if both clauses are satisfied. With a $2k$ set instance we have at least $k$ sets split if and only if we have at least $3k$ clauses satisfied. With our kernel, this direct approach would be better than the method described in this paper if the Max Sat running time could be improved below $\mathcal{O}(2^{m/3})$, where $m$ is the number of clauses.

We would like to acknowledge Daniel Kral for insightful remarks.

# BIBLIOGRAPHY


[AE97] G. Andersson and L. Engebretsen, Better approximation algorithms for set splitting and Not-All-Equal-Sat, *Information Processing Letters*, **65**(1988) 305–311.

[CK04] J. Chen, I. Kanj, Improved Exact Algorithms for Max-Sat, *Discrete Applied Mathematics* **142**(2004), 17–27.

[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, in *Proceedings of WG2004*, *LNCS* (2004).

[DFR03] F. Dehne, M. Fellows, and F. Rosamond, An FPT Algorithm for Set Splitting, in *Proceedings WG2004 - 30th Workshop on Graph Theoretic Concepts in Computer science*,LNCS 2004.

[DFRS04] F. Dehne, M. Fellows, F. Rosamond, and P. Shaw, Greedy Localization, Iterative Compression and Modeled Crown Reductions: New FPT Techniques and Improved Algorithms for Max Set Splitting and Vertex Cover, *Proceedings of IWPEC04* LNCS **3162**(2004), 271–281.

[FK02] S. Fedin and A. Kulikov, A $2^{|E|/4}$-time Algorithm for MAX-CUT. *Zapiski nauchnyh seminarov POMI*, **293** (2002), 129–138. English translation to appear in Journal of Mathematical Sciences.

[FHRST04] M.Fellows, P.Heggernes, F.Rosamond, C. Sloper, J.A.Telle, Finding $k$ disjoint triangles in an arbitrary graph, *To appear WG2004*

[GJ79] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* (W.H. Freeman, San Francisco, 1979).

[MR99] M. Mahajan, V. Raman, Parameterizing above guaranteed values: MaxSat and MaxCut, *Journal of Algorithms* **31**(1999), 335–354.

[P04] E. Prieto, The Method of Extremal Structure on the $k$-Maximum Cut Problem. *Manuscript, to appear*

[PS04] E.Prieto and C. Sloper, Reducing to Independent Set Structure — the Case of $k$-INTERNAL SPANNING TREE. *To appear*

[Pe94] E. Petrank, The hardness of approximation: Gap location, *Computational Complexity*, **4**(1994), 133–157.

[W03] G. Woeginger, Exact Algorithms for NP-Hard Problems. A Survey, in *Proceedings of 5th International Workshop on Combinatorial Optimization-Eureka, You Shrink!* Papers dictated to Jack Edmonds, M. Junger, G. Reinelt, and G. Rinaldi (Festschrift Eds.) LNCS **2570** (2003), pp. 184–207.

[ZL01] H. Zhang and C.X.Ling, An improved learning algorithm for augmented naive Bayes, *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, LNCS **2035**(2001), pp.581–586.