Towards a Taxonomy of Techniques for Designing Parameterized Algorithms

Christian Sloper and Jan Arne Telle

Department of Informatics, University of Bergen, Norway
{sloper, telle}@ii.uib.no

Abstract. A survey is given of the main techniques in parameterized algorithm design, with a focus on formal descriptions of the less familiar techniques. A taxonomy of techniques is proposed, under the four main headings of Branching, Kernelization, Induction and Win/Win. In this classification the Extremal Method is viewed as the natural maximization counterpart of Iterative Compression, under the heading of Induction. The formal description given of Greedy Localization generalizes the application of this technique to a larger class of problems.

1 Introduction

The field of parameterized algorithms continues to grow. It is a sign of its success that in the literature today there exists over twenty differently named techniques for designing parameterized algorithms[1]. Many parameterized algorithms build on the same ideas, and as a problem solver it is important to be familiar with these general themes and ideas. Several survey articles [F03,DFRS04] and books [N02,DF99] have identified common themes like Bounded Search Trees, Kernelization and Win/Win. In this paper we make an attempt at a full taxonomy encompassing all known techniques. In addition to its pedagogic value, we believe that such a taxonomy could help in developing both new techniques and new combinations of known techniques.

We classify the known techniques under the four main themes of Branching, Kernelization, Induction and Win/Win, see Figure 1. The four main themes are described in separate sections. Each technique is introduced, possibly with an example, and its placement in the taxonomy is argued for. In this extended abstract we focus on the most novel aspects, *e.g.* the grouping of Greedy Localization and Color Coding together with Bounded Search Trees under the heading of Branching algorithms, and the placement of the Extremal Method as the maximization counterpart of Iterative Compression under the heading of

[1] This includes Bounded Search Trees [DF99], Data Reduction [N02], Kernelization [DF99], The Extremal Method [FMRS00], The Algorithmic Method [P05], Catalytic Vertices [FMRS00], Crown Reductions [CFJ04], Modelled Crown Reductions [DFRS04], Either/Or [PS03], Reduction to Independent Set Structure [PS05], Greedy Localization [DFRS04], Win/Win [F03], Iterative Compression [DFRS04], Well-Quasi-Ordering [DF99], FPT through Treewidth [DF99], Search Trees [N02], Bounded Integer Linear Programming [N02], Color Coding [AYZ95], Method of Testsets [DF99], Interleaving [NR00]

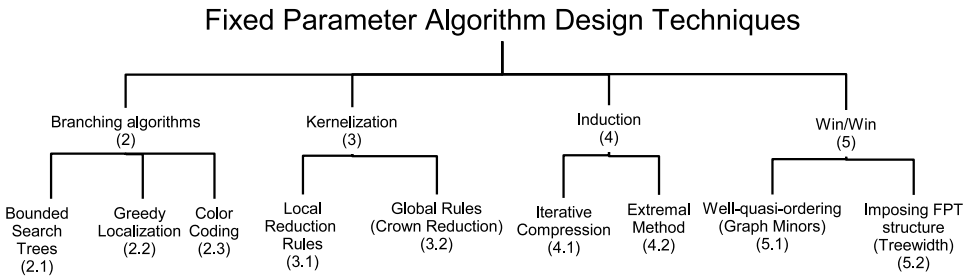

Fig. 1. The first two levels of the taxonomy, labelled by section number.

Induction. We give a generic and formal description of Greedy Localization that encompasses problems hitherto not known to be FPT solvable by this technique.

Sometimes a technique is known under different names[2] or it is a variant of a more general technique[3]. Clearly, the fastest FPT algorithm for a problem will usually combine several techniques. In some cases such a combination has itself been given a separate name[4]. For simplicity and lack of space this extended abstract does not consider all these variations of the most general techniques, and our discussion is restricted to the first two levels of the hierarchy in Figure 1. A note on notation: the chosen parameter for a problem is usually not stated explicitly, instead we use the convention that the variable name k always denotes the parameter.

2 Branching Algorithms

We start by considering algorithm techniques that use a branching strategy to create a set of subproblems such that at least one of the subproblems is a yes-instance if and only if the original problem is a yes-instance. Techniques that create branching algorithms are Bounded Search Trees, Greedy Localization and Color Coding. For a branching algorithm to have FPT running time it suffices to require that it (see [S06] for a proof):

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

[2] This is the case with Data Reduction=Local Reduction Rules, Either/Or=Win/Win, Search Trees=Bounded Search Trees and other names like Hashing [DF99]=Color Coding which do not seem to be in use anymore

[3] This is the case with Catalytic Vertices $\subseteq$ Local Reduction Rules, The Algorithmic Method $\subseteq$ Extremal Method, Modelled Crown Reductions $\subseteq$ Crown reductions, FPT by Treewidth $\subseteq$ Imposing FPT structure, Reduction to Independent set structure $\subseteq$ Imposing FPT structure, Method of Testsets $\subseteq$ Imposing FPT structure.

[4] This is the case with Interleaving which combines Bounded Search Trees and Local Reduction Rules. The technique known as Bounded Integer Linear Programming can in a similar way be viewed as a combination of a Branching algorithm with Local Reduction Rules.

2.1 Bounded Search Trees

The method of Bounded Search Trees is arguably the most famous parameterized algorithm design technique and good general descriptions of it are plentiful, see for example the books [DF99,N02]. For lack of space we describe it very simply by saying that it is a branching algorithm which is strictly recursive.

2.2 Greedy Localization

Greedy Localization is a technique that uses a clever first branching to start off the recursive search for the solution. It was introduced in a paper that has since appeared in Journal of Algorithms [JZC04] and popularized in an IWPEC'04 paper [DFRS04]. Our aim is to show that if a parameterized problem satisfies the following conditions 1 and 2 then Greedy Localization will give an FPT algorithm.

1. The problem can be formulated as that of finding k pairwise non-overlapping 'objects' in an input instance G, with objects being special subsets of size depending only on k of a ground set W of G.
2. For any $R \subseteq W$ and $X \subseteq W$ we can decide in FPT time if there exists $S \subseteq W \setminus X$ such that $S \cup R$ is an object.

Not all bounded-size subsets of W are objects, and an obvious requirement for the problem to have an FPT algorithm is that for any $R \subseteq W$ we must be able to decide in FPT time if R is an object or not. Condition 2 can be seen as a strengthening of this obvious requirement and we will refer to S as an 'extension' of the 'partial object' R to a 'full object' $R \cup S$ avoiding X.

The figure on the next page gives the Greedy Localization algorithm, in non-deterministic style, for a problem satisfying these two conditions. It uses the notation that for a set of partial objects $B = \{B_1, B_2, \ldots, B_k\}$ the ground elements contained in B are denoted by $W_B = \bigcup_{B_i \in B} B_i$.

Theorem 1. *If a parameterized problem satisfies conditions 1 and 2 above then the algorithm 'GREEDY LOCALIZATION' is an FPT algorithm for this problem.*

Proof. The algorithm starts by computing an inclusion maximal non-overlapping set of objects A. By condition 2 this first step can be done in FPT time as follows: repeatedly extend the emptyset to a full object while avoiding a set X, by calling subroutine EXTEND$(\emptyset, X)$ with X initially empty, and adding the extension to X before the next iteration. When no extension exists we are assured that the sequence of extensions found must be an inclusion maximal non-overlapping set of objects A.

The crucial aspect that makes the algorithm correct is that if A and B are two inclusion maximal non-overlapping sets of objects then for any object B_i in B there is an object A_j in A such that A_j and B_i overlap, since otherwise A is not maximal. Thus, if the instance contains such a set B of at least k objects,

Algorithm GREEDY LOCALIZATION /* non-deterministic */
Input: Instance G with ground set W, and an integer k
Output: Yes if G has k non-overlapping objects, otherwise No
compute an inclusion maximal non-overlapping set of objects A
if A contains at least k objects then **halt** and **answer** 'Yes'
else if $|W_A| < k$ then **halt** and **answer** 'No'
else guess $\{v_1, v_2, \ldots, v_k\} \subseteq W_A$ and let B_i be partial object containing v_i $(1 \leq i \leq k)$
BRANCHING($B = \{B_1, B_2, \ldots, B_k\}$)

Subroutine BRANCHING
Input: Set of partial objects $B = \{B_1, B_2, \ldots, B_k\}$
Output: Yes if B can be extended to a set of full objects, otherwise No
$F = S = \emptyset$
for $j = 1$ to k
  if B_j not full then {
    $S =$EXTEND($B_j, W_B \cup F$)
    if $S == \emptyset$ then **break**
    else $F = F \cup S$
    }
if all objects could be extended then **halt** and **answer** 'Yes'
else if $F == \emptyset$ then **halt** and **answer** 'No'
else guess $v \in F$ and add v to B_j /* j is value of parameter at break */
BRANCHING($B = \{B_1, B_2, \ldots, B_k\}$)

Subroutine EXTEND
Input: Partial object B_i, unavailable ground elements X
Output: Ground elements $S \subseteq W \setminus X$ whose addition to B_i gives a full object or
$S = \emptyset$ if this is not possible

then we can guess k ground elements appearing in A, with A constructed in the first step of the algorithm, such that these ground elements belong to k separate objects of B (and if $|W_A| < k$ we can answer 'No'.) The Branching subroutine is called on these k one-element partial objects and we greedily try to extend them to full objects. If this fails for some object B_j, after having added extension elements F to objects $B_1, B_2, ..., B_{j-1}$, then there must exist an element v from F that should have instead been used to extend B_j. We then simply guess the element v and try again.

For a deterministic algorithm the guesses are replaced by branchings, and we give a No answer iff all branches answer No. The first branching is of size $\binom{|W_A|}{k}$, the remainder of the branches are of size $|F|$, and the total height of the tree is bounded by k times the maximum size of an object since at each level one new ground element is added to V_B. All these are bounded by a function depending on k as we assumed in condition 1 that each object had size depending on k only. The calls to the Extend subroutine are FPT by condition 2. Hence the algorithm is FPT. □

For example, this implies that deciding if a graph G contains k vertex-disjoint cycles on k vertices is FPT by Greedy Localization. The ground set W will be the vertex set of G and the objects will be subsets of k vertices inducing a subgraph containing a k-cycle, to satisfy condition 1. Given $R, X \subseteq W$ we can by an FPT algorithm designed using the Color Coding technique, see the next section, decide if there exists $S \subseteq W \setminus X$ such that $R \cup S$ induces a subgraph containing a k-cycle, to satisfy condition 2. By Theorem 1 the Greedy Localization meta-algorithm therefore solves the problem in FPT time. For packing of edge-disjoint cycles a similar argument holds, with W being the edge set of the graph.

2.3 Color Coding

Color Coding is a technique that was introduced by Alon, Yuster, and Zwick in their paper 'Color Coding' [AYZ95] and is characterized by a powerful first branching step. 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 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 color classes. 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. The color coding technique gives an FPT algorithm whenever this colored subproblem can be solved in FPT time. Perhaps the strongest results using Color Coding are obtained in [FKNRSTW04] where it is combined with kernelization to give FPT algorithms for a large variety of problems.

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 could be slower than, for example, a $2^{k \log k}$ algorithm obtained through other techniques.

3 Kernelization

Under the heading of kernelization we combine techniques that reduce a general instance into an equivalent *kernel*, i.e. , an instance whose total size is bounded

by a function depending only on the parameter. We distinguish between *local reductions* and *global reductions*.

3.1 Local Reductions

Local reduction is a well known technique. We say that a local *reduction rule* is a rule that identifies a certain constant-size structure LHS in the instance (the left-hand-side) and modifies it to RHS (the right-hand-side). This must be done in such a way that the original instance (G, k) has a positive solution iff the reduced instance (G', k') has one. The goal is to find a set of rules such that repeatedly applying the rules to an instance will either determine the answer directly or give a kernel.

3.2 Global Reduction Rules - Crown Reduction

Lately there has been a focus on reduction rules that do not follow the pattern of finding a local structure of constant size. In this section we describe reduction rules based on finding *crown decompositions* in graphs.

Definition 2. *A* crown decomposition *of a graph $G = (V, E)$ is a partitioning of V into sets C, H, R, where C and H are both nonempty, such that:*

1. *C is an independent set.*
2. *There is 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 ab is an edge. We call ab a matched edge if $m(a) = b$.*

When using a crown decomposition (C, H, R) in a reduction rule for a graph G we must show that we can can remove or modify $(C \cup H)$ to obtain a reduced instance (G', k') which is a Yes-instance if and only if (G, k) is a Yes-instance. For example, it is easy to see that G has a Vertex Cover of size k iff the graph G' resulting from removing $C \cup H$ has a Vertex Cover of size $k - |H|$. Usually more complicated reduced instances and arguments are necessary. For example, an FPT algorithm for k-Internal Spanning Tree [PS05] uses crown reduction rules that remove only the vertices of C not incident to a matched edge.

Although it is possible to determine if a graph has a crown decomposition in polynomial time [ACFL04], this technique is often combined with the following lemma by Chor, Fellows, and Juedes [CFJ04]

Lemma 1. *If a graph $G = (V, E)$ has an independent set I such that $|N(I)| < |I|$, then a crown decomposition (C, H, R) for G such that $C \subseteq I$ can be found in time $\mathcal{O}(|V| + |E|)$.*

The notation $N(I)$ denotes vertices in $V \setminus I$ that are adjacent to a vertex in I. Since it is W[1]-hard to find a large independent set we cannot directly apply Lemma 1. To see how the Lemma can be used, consider k-Vertex Cover on a graph G. We first compute a maximal matching M in G, and let V_M be

the vertices incident to the edges in M. If $|V_M| > 2k$ then there does not exist a Vertex Cover for G of size k. If $|V_M| \leq 2k$ and $|V \setminus V_M| \leq 2k$ then G is a kernel. Otherwise, $|V_M| \leq 2k$ and $|V \setminus V_M| > 2k$ and since $V \setminus V_M$ is an independent set with $|N(V \setminus V_M)| \leq |V_M| \leq 2k < |V \setminus V_M|$ we have, by Lemma 1, a crown decomposition of G, that can be used to reduce the graph as described above. Repeating this process gives an FPT algorithm for k-Vertex Cover.

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 problems (SAT). An *autarky* is a partial truth assignment (assigning true/false to only a subset of the variables) such that each clause that contains a variable 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 the satisfiability problem. It is easy to see that the matching autarky is a crown decomposition in this bipartite graph. The main protagonist in this field is Oliver Kullmann [K00,K03], who has developed an extensive theory on different types of autarkies.

4 FPT by Induction

We discuss techniques closely related to mathematical induction. 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.

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 try to construct a solution based on the inductive assumption that we have a solution to smaller problems. For example, this leads to the well known Insertion Sort algorithm by noting that we can sort sequences of n elements by first sorting $n-1$ elements and then inserting the last element at its correct place.

This inductive technique may also be applied to the design of FPT algorithms but more care must be taken on two accounts: (a) we have one or more parameters and (b) we are dealing with decision problems. 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?

We first 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.

4.1 For Minimization - Iterative Compression

In this section we present Iterative Compression which works well on certain parameterized minimization problems. Let us assume that we can inductively (recursively) compute the solution for the smaller instance $(G - v, k)$. Since our problems are decision problems, we get either a 'Yes'-answer or a 'No'-answer. In both cases we must use the information provided by the answer to compute the solution for (G, k). We must assume that for a 'Yes'-instance we also have a certificate that verifies that the instance is a 'Yes'-instance and it is this certificate that must be used to compute the solution for (G, k). However, for a 'No'-answer we may receive no extra information. A class of problems where 'No'-answers carry sufficient information is the class of *monotone* problems in which the 'No'-instances are closed under element addition. Thus if a problem is monotone we can immediately answer 'No' for (G, k) whenever $(G - v, k)$ is a 'No'-instance.

Two papers that use this type of induction on monotone graph minimization problems are [RSV03] which shows that k-Odd Cycle Cover (is it possible to delete k vertices from G to obtain a bipartite graph) is FPT, and [DFRS04] where a $2k$ kernel is given for k-Vertex Cover without using the complicated Nemhauser-Trotter results [NT75].

Note that many minimization problems are not monotone, like Dominating Set where the addition of a universal vertex always changes a 'No' answer to 'Yes' (unless $k \geq n$). For such problems we believe that Iterative Compression is ill suited.

4.2 For Maximization - The Extremal Method

For maximization problems we consider smaller instances of the type $(G, k-1)$, and induct on k instead of n. We say that a problem is *parameter monotone* if the 'No'-instances are closed under parameter increment, *i.e.* if instance (G, k) is a 'No'-instance then (G, k') is also a 'No'-instance for all $k' > k$.

The *Method of Extremal Structure*[5] is a design technique that works well for parameter monotone maximization problems. In this technique we do not focus on any particular instance (G, k), but instead investigate the structure of graphs that are 'Yes'-instances for k, but 'No'-instances for $k+1$. Let $\mathcal{G}(k)$ be the class of such graphs, i.e., $\mathcal{G}(k) = \{G \mid (G, k)$ is a 'Yes'-instance, and $(G, k+1)$ is a 'No'-instance $\}$.

Our ultimate goal is to prove that there exists a function $f(k)$ such that $h \max\{|V(G)| \mid G \in \mathcal{G}(k)\} \leq f(k)$. This is usually not possible without some refinement of $\mathcal{G}(k)$, to do this we make a set of observations E of the following type:

[5] An exposition of this design technique can be found in E. Prieto's PhD thesis [P05]

Since (G, k) is a 'Yes'-instance, but $(G, k+1)$ is a 'No'-instance, G has property p. (1)

Given a set of such observations E and consequently a set of properties P we try to devise a set of reduction rules R that apply specifically to large graphs having the properties P. We call our refined class $\mathcal{G}_R(k) = \{G \mid$ no reduction rule in R applies to (G, k), and (G, k) is a 'Yes'-instance, and $(G, k+1)$ is a 'No'-instance $\}$. If we can add enough observations to E and reductions rules to R to prove that there is a function $f(k)$ such that $\max\{|V(G)| \mid G \in \mathcal{G}_R(k)\} \leq f(k)$ we have proven that:

If i) no rule in R applies to (G, k) and ii) (G, k) is a 'Yes'-instance and iii) $(G, k+1)$ 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 a *kernelization* lemma follows, saying that 'If no rule in R applies to (G, k) and $|V(G)| > f(k)$, then (G, k) is a 'Yes'-instance.

It is not immediately obvious that this can be viewed as an inductive process, but we will now make this clear by presenting the *Algorithmic Method*, a version of the 'Extremal Method'. Here the 'Extremal Method' can be used as the inductive step, going from k to $k+1$, in an inductive algorithm.

As its base case, the algorithm decides $(G, 0)$, which is usually a trivial 'Yes'-instance for a maximization problem. Our induction hypothesis is that we can decide (G, k'). Then as long as $k' + 1 \leq k$ we try to compute $(G, k'+1)$. If (G, k') is a 'No'-instance we can immediately answer 'No' for $(G, k'+1)$ as the problem is parameter monotone. Otherwise we can now make an algorithmic use of observations of the type defined for extremal method ((1) above). For each of the properties $p \in P$ we check if G has the property p. If G does not have property p then since (G, k') is a 'Yes'-instance it follows that $(G, k'+1)$ is also a 'Yes'-instance. By the same reductions and observations (although the reader should observe that we here require properties to be FPT time verifiable), we obtain that

If no observation in E or reduction rule R applies to $(G, k'+1)$ then $|V(G)| < f(k)$.

At that point we can invoke a brute force algorithm to obtain either a solution S or a 'No'-answer for $(G, k'+1)$. This answer for $(G, k'+1)$ can then be used in the next step, $k'+2$, of our inductive algorithm.

This technique, either the 'Extremal Method' or its variant the 'Algorithmic Method', can be applied successfully to a range of problems, such as: k-Max Cut [P04], k-Leaf Spanning Tree [P05], k-Non-Blocker [P05], k-Edge-disjoint Triangle-Packing [MPS04], k-$K_{1,s}$-packing [PS04], k-K_3-packing [FHRST04], k-Set Splitting [DFR03], and k-Internal Spanning Tree [PS05].

5 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 section we focus on techniques exploting this behavior. According to [DH05] the only known algorithms with sub-exponential running time $\mathcal{O}^*(c^{\sqrt{k}})$ are algorithms based on imposing treewidth and branchwidth structure on the complicated cases, and these fall into the Win/Win category.

5.1 Well-quasi-ordering and Graph Minors

Robertson and Seymour have shown that i) the set of finite graphs are well-quasi-ordered under minors and ii) the H-Minor problem that checks if H is a minor of some input graph, with $k = |V(H)|$, is FPT. These two facts suffice to prove that any parameterized graph problem whose Yes-instances (or No-instances) are closed under minors is FPT. If analogous structural results could be shown for some other relation, besides minors, then for problems closed under this other relation we would also get FPT algorithms. Thus the general technique is called 'well-quasi-ordering'. We consider this a Win/Win algorithm as we relate the problem we wish to solve to the FPT problem of checking if one of the forbidden minors (or whatever other relation is involved) appear in our problem instance.

Let us briefly explain the main ideas. A well-quasi-ordering is a reflexive and transitive ordering which has no infinite antichain, meaning that any set of elements no two of which are comparable in the ordering must be finite. 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 contracting edges of a subgraph of G. The Graph Minors Theorem [RS99] states that 'The set of graphs are well-quasi-ordered by the minor relation'. Combined with H-minor testing this can be used to prove existence of an FPT algorithm for any problem A with the property that for any k the 'Yes'-instances are closed under minors. In other words, let us assume that if A_k is the class of graphs G such that (G, k) is a 'Yes'-instance to problem A and $H \preceq_m G$ for some $G \in A_k$ then $H \in A_k$ as well. Consider the Minimal Forbidden Minors of A_k, denoted $MFM(A_k)$, defined as follows: $MFM(A_k) = \{G \mid G \notin A_k \text{ and } (\forall H \preceq_m G, H \in A_k \vee H = G)\}$.

By definition, $MFM(A_k)$ is an antichain of the $\preceq_m$ ordering of graphs so by the Graph Minors Theorem it is finite. Beware that the non-constructive nature of the proof of the Graph Minors Theorem implies that we can in general not construct the set $MFM(A_k)$ and thus we can only argue for the existence of an FPT algorithm. We do this by noting that (G, k) is a Yes-instance of problem A iff there is no $H \in MFM(A_k)$ such that $H \preceq_m G$. Since $MFM(A_k)$ is independent of $|G|$ we can therefore decide if (G, k) is a Yes-instance in FPT time by $|MFM(A_k)|$ calls of H-Minor. Armed with this powerful tool, all we have to do to prove that a parameterized graph problem is FPT is to show

that the Yes-instances are closed under the operations of edge contraction, edge deletion and vertex deletion.

5.2 Imposing FPT structure and Bounded treewidth

In the literature on parameterized graph algorithms there are several notable occurrences of a Win/Win strategy that imposes a tree-like structure on the class of problematic graphs, in particular by showing that they must have treewidth bounded by a function of the parameter. This is then combined with the fact that many NP-hard problems are solvable in FPT time if the parameter is the treewidth of the input graph.

Let us briefly explain this technique in the case of finding k-dominating sets in planar graphs, where a very low treewidth bound on Yes-instances gives very fast FPT algorithms. In [ABFKN02] it is shown that a planar graph that has a k-dominating set has treewidth at most $c\sqrt{k}$ [ST94], for an appropriate constant c. Thus we have a win/win relationship, since we can check in polynomial time if a planar graph has treewidth at most $c'\sqrt{k}$, for some slightly larger constant c', and if so find a tree-decomposition of this width. If the treewidth is higher we can safely reject the instance, and otherwise we can run a dynamic programming algorithm on its tree-decomposition, parameterized by $c'\sqrt{k}$, to find in FPT time the optimal solution. In total this gives a $\mathcal{O}^*(c''^{\sqrt{k}})$ algorithm for deciding if a planar graph has a dominating set of size k.

A series of papers have lowered the constant c'' of this algorithm, by several techniques, like moving to branchwidth instead of treewidth, by improving the constant c, and by improving the FPT runtime of the dynamic programming stage. Yet another series of papers have generalized these 'subexponential in k' FPT algorithms from dominating set to all so-called bidimensional parameters and also from planar graphs to all graphs not having a fixed graph H as minor [DH05].

6 Conclusion

We believe that a taxonomy of techniques for designing parameterized algorithms will invariably develop over time and in this paper we made a comprehensive attempt. Many such classification schemes are possible, and the one proposed in this paper is the result of many discussions, primarily between the two authors, but also with other people in the field, see [S06]. Given the nature of such classifications and the continual development of the field we expect that this proposal will be criticized and altered over time, but it is our hope that it will lay the ground for fruitful discussions.

References

[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* 3353, p 235-244(2004).

[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.

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

[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.

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

[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. Springer Verlag, 2004.

[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

[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.

[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.

[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).

[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.

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

[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.

[MPS04] L. Mathieson, E. Prieto, P. Shaw. Packing Edge Disjoint Triangles: A Parameterized View. *Proceedings of IWPEC 04*, 2004, LNCS 3162,pages 127-137.

[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. Manuscript, 2002, a book version was recently announced by Oxford University Press.

[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.

[P04] E. Prieto. The Method of Extremal Structure on the k-Maximum Cut Problem. *Proc. Computing: The Australasian Theory Symposium (CATS 2005). Conferences in Research and Practice in Information Technology*, 2005, vol. 41, pages 119-126.

[P05] E. Prieto. Systematic kernelization in FPT algorithm design. *PhD thesis, University of Newcastle, Australia* 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 04)*, LNCS vol. 3162, pages 138-149, 2004.

[PS05] E. Prieto, C. Sloper. Reducing to Independent Set Structure — the Case of k-Internal Spanning Tree', *Nordic Journal of Computing*, 2005, vol 12, nr 3, pp. 308-318.

[RS99] N. Robertson, PD. Seymour. 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.

[S06] C.Sloper. Techniques in Parameterized Algorithm Design. *PhD thesis, University of Bergen*, 2006, (http://www.ii.uib.no/ sloper).

[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.

[ST94] P. D. Seymour and R. Thomas. Call routing and the ratcatcher. *Combinatorica*, vol 14(2), pages 217 – 241, 1994.