AGAPE 2009
Notes on Parameterized Complexity

Mike Fellows

University of Newcastle, Australia
michael.fellows@cs.newcastle.edu.au

Abstract. These notes cover two presentations:
(1) A general overview and introduction to the field.
(2) Parameterized intractability and complexity classes.

1 Introduction to Parameterized Complexity

1.1 Two Forms of Fixed-Parameter Complexity

Many natural computational problems are defined on input consisting of various information, for example, many graph problems are defined as having input consisting of a graph $G = (V, E)$ and a positive integer k. Consider the following well-known problems:

VERTEX COVER
Input: A graph $G = (V, E)$ and a positive integer k.
Question: Does G have a vertex cover of size at most k? (A *vertex cover* is a set of vertices $V' \subseteq V$ such that for every edge $uv \in E$, $u \in V'$ or $v \in V'$ (or both).)

DOMINATING SET
Input: A graph $G = (V, E)$ and a positive integer k.
Question: Does G have a dominating set of size at most k? (A *dominating set* is a set of vertices $V' \subseteq V$ such that $\forall u \in V$: $u \in N[v]$ for some $v \in V'$.)

Although both problems are *NP*-complete, the input *parameter* k contributes to the complexity of these two problems in two qualitatively different ways.

1. There is a simple *bounded search tree* algorithm for VERTEX COVER that runs in time $O(2^k n)$
2. The best known algorithm for DOMINATING SET is basically just the brute force algorithm of trying all k-subsets. For a graph on n vertices this approach has a running time of $O(n^{k+1})$.

(Easy) Exercise: What is the search tree algorithm for VERTEX COVER refered to above?

The table below shows the contrast between these two kinds of complexity.

In these two example problems, the parameter is the size of the solution being sought. But a parameter that affects the compexity of a problem can be many things.

	$n = 50$	$n = 100$	$n = 150$
$k = 2$	625	2,500	5,625
$k = 3$	15,625	125,000	421,875
$k = 5$	390,625	6,250,000	31,640,625
$k = 10$	1.9×10^{12}	9.8×10^{14}	3.7×10^{16}
$k = 20$	1.8×10^{26}	9.5×10^{31}	2.1×10^{35}

Table 1. The Ratio $\frac{n^{k+1}}{2^k n}$ for Various Values of n and k.

Example. *The nesting depth of a logical expression.* ML is a logic-based programming language for which relatively efficient compilers exist. One of the problems the compiler must solve is the checking of the compatibility of type declarations. This problem is known to be complete for *EXP* (deterministic exponential time) [HM91], so the situation appears discouraging from the standpoint of classical complexity theory. However, the implementations work well in practice because the ML Type Checking problem is *FPT* with a running time of $O(2^k n)$, where n is the size of the program and k is the maximum nesting depth of the type declarations [LP85]. Since normally $k \leq 5$, the algorithm is clearly practical on the natural input distribution.

The parameter can be size of the solution, or some structural aspect of the natural input distribution — and many other things (to be discussed below).

In the favorable situations (as for Vertex Cover and Type Checking in ML), the exponential cost of solving the problem (that is expected, since the problems are NP-hard) can be entirely confined to an exponential function of the *parameter*, with the overall input size n contributing polynomially.

1.2 Clashes of Function Classes; Multivariate Complexity and Algorithmics

The familiar "P versus NP" framework, that we call the *classical framework*, is fundamentally centered on the notion of *polynomial time*, and this is a one-dimensional framework: there is one measurement (or variable) at work, the overall input size n.

The classical framework revolves around a contrast between two function classes: the *good* class of running times of algorithms of the form: $O(n^c)$, time that is polynomial in the one measurement n. The *bad* class of run times is those of the form 2^{n^c}, and the drama concerns methods for establishing that concrete problems admit good algorithms (and if so, maybe better algorithms?), the *positive toolkit*, or if they only admit bad algorithms (modulo reasonable conjectures), the *negative toolkit* that in the classical case is about NP-hardness, EXP-hardness, etc.

Worth noting at this point is that one of the main motivations to parameterized complexity (and many other approaches) is that while the classical theory is beautiful, and a handful of important problems are in P, the vast majority of problems have turned out to be NP-hard or worse.

Parameterized complexity is basically a two-dimensional sequel, based similarly on a contrast between two function classes in a two-dimensional setting, where in addition to the overall input size n, we have a second measurement (or variable) that captures something else significant that affects computational complexity (and the opportunities for efficient algorithm design), the parameter k (that might be solution size, or something structural about typical inputs, ... or many other things).

How do we formalize this?

Definition 1. *A parameterized language L is a subset $L \subseteq \Sigma^* \times \Sigma^*$. If L is a parameterized language and $(x, k) \in L$ then we will refer to x as the* main part*, and refer to k as the* parameter*.*

A parameter may be non-numerical, and it can also represent an aggregate of various parts or structural properties of the input.

Definition 2. *A parameterized language L is* multiplicatively fixed-parameter tractable *if it can be determined in time $f(k)q(n)$ whether $(x, k) \in L$, where $|x| = n$, $q(n)$ is a polynomial in n, and f is a function (unrestricted).*

Definition 3. *A parameterized language L is* additively fixed-parameter tractable *if it can be determined in time $f(k) + q(n)$ whether $(x, k) \in L$, where $|x| = n$, $q(n)$ is a polynomial in n, and f is a function (unrestricted).*

(Easy) Exercise. Show that a parameterized language is additively fixed-parameter tractable if and only if it is multiplicatively fixed-parameter tractable. This emphasizes how cleanly fixed-parameter tractability isolates the computational difficulty in the complexity contribution of the parameter.

The following definition provides us with a place to put all those problems that are "solvable in polynomial time for fixed k" without making the central distinction about whether this "fixed k" is ending up in the exponent or not (as with the brute force algorithm for Dominating Set).

Definition 4. *A parameterized language L belongs to the class XP if it can be determined in time $f(k)n^{g(k)}$ whether $(x, k) \in L$, where $|x| = n$, with f and g being unrestricted functions.*

Is it possible that $FPT = XP$? This is one of the few structural questions concerning parameterized complexity that currently has an answer [DF98].

Theorem 1. *FPT is a proper subset of XP.*

Summarizing the main point: parameterized complexity is about a natural bivariate generalization of the P versus NP drama. This inevitably leads to two toolkits: the *positive toolkit* of FPT methods (that Daniel Marx will lecture about), and the *negative toolkit* that basically provides a parameterized analog of Cook's Theorem, and methods for showing when fixed-parameter tractable algorithms for parameterized problems are not possible (modulo reasonable assumptions).

There is a larger context captured by the reasonable question that is often asked:

If Parameterized Complexity is the natural two-dimensional sequel to P versus NP, then what is the three-dimensional sequel?

Nobody currently knows the answer. Ideally, one would like to have a *fully multivariate* perspective on complexity analysis and algorithm design that meets the following criteria:
• In dimension 1, you get the (basic) P versus NP drama.
• In dimension 2, you get the (productive) FPT versus XP drama.
• In all dimensions, you have concrete problems where the contrasting outcomes are natural and consequential, and the theory is routinely doable.

Open research problem. Is there such a fully multivariate mathematical perspective?

Introducing (at least one) secondary variable k beyond the overall input size n allows us to ask many new and interesting questions that cannot be asked in any mathematically natural way in the classical framework. Much of this interesting traction is based on the various ways that parameterization can be deployed.

1.3 Parameters Can Be Many Things

There are many ways that parameters arise naturally, for example:

• *The size of a database query.* Normally the size of the database is huge, but frequently queries are small. If n is the size of a relational database, and k is the size of the query, then answering the query (MODEL CHECKING) can be solved trivially in time $O(n^k)$. It is known that this problem is unlikely to be *FPT* [DFT96,PY97] because it is hard for $W[1]$, the parameterized analog of NP-hardness. However, if the parameter is the size of the query and the treewidth of the database, then the problem is fixed-parameter tractable [Gr01b].

• *The number of species in an evolutionary tree.* Frequently this parameter is in a range of $k \leq 50$. The PHYLIP computational biology server includes an algorithm which solves the STEINER PROBLEM IN HYPERCUBES in order to compute possible evolutionary trees based on (binary) character information. The exponential heuristic algorithm that is used is in fact an *FPT* algorithm when the parameter is the number of species.

• *The number of variables or clauses in a logical formula, or the number of steps in a deductive procedure.* Determining whether at least k clauses of a CNF formula F are satisfiable is *FPT* with a running time of $O(|F|+1.381^k k^2)$ [BR99]. Since at least half of the m clauses of F can always be satisfied, a more natural parameterization is to ask if at least $m/2 + k$ clauses can be satisfied — this is *FPT* with a running time of $O(|F| + 6.92^k k^2)$ [BR99]. Implementations indicate that these algorithms are quite practical [GN00].

• *The number of moves in a game, or the number of steps in a planning problem.* While most game problems are *PSPACE*-complete classically, it is known that some are *FPT* and others are likely not to be *FPT* (because they are hard for $W[1]$), when parameterized by the number of moves of a winning strategy

[ADF95]. The size n of the input game description usually governs the number of possible moves at any step, so there is a trivial $O(n^k)$ algorithm that just examines the k-step game trees exhaustively.

• *The number of facilities to be located.* Determining whether a planar graph has a dominating set of size at most k is fixed-parameter tractable by an algorithm with a running time of $O(8^k n)$ based on kernelization and search trees. By different methods, an *FPT* running time of $O(3^{36\sqrt{k}})n$ can also be proved.

• *An unrelated parameter.* The input to a problem might come with "extra information" because of the way the input arises. For example, we might be presented with an input graph G together with a k-vertex dominating set in G, and be required to compute an optimal bandwidth layout. Whether this problem is *FPT* is open.

• *The amount of "dirt" in the input or output for a problem.* In the Maximum Agreement Subtree (MAST) problem we are presented with a collection of evolutionary trees trees for a set X of species. These might be obtained by studying different gene families, for example. Because of errors in the data, the trees might not be isomorphic, and the problem is to compute the largest possible subtree on which they do agree. Parameterized by the number of species that need to be deleted to achieve agreement, the MAST problem is *FPT* by an algorithm having a running time of $O(2.27^k + rn^3)$ where r is the number of trees and n is the number of species [NR01].

• *The "robustness" of a solution to a problem, or the distance to a solution.* For example, given a solution of the Minimum Spanning Tree problem in an edge-weighted graph, we can ask if the cost of the solution is robust under all increases in the edge costs, where the parameter is the total amount of cost increases.

• *The distance to an improved solution.* Local search is a mainstay of heuristic algorithm design. The basic idea is that one maintains a *current solution*, and iterates the process of moving to a neighboring "better" solution. A neighboring solution is usually defined as one that is a single step away according to some small edit operation between solutions. The following problem is completely general for these situations, and could potentially provide a valuable subroutine for "speeding up" local search:

k-Speed Up for Local Search
Input: A solution S, k.
Parameter: k
Output: The best solution S' that is within k edit operations of S.

• *The goodness of an approximation.* If we consider the problem of producing solutions whose value is within a factor of $(1+\epsilon)$ of optimal, then we are immediately confronted with a natural parameter $k = 1/\epsilon$. Many of the recent PTAS results for various problems have running times with $1/\epsilon$ in the exponent of the polynomial. Since polynomial exponents larger than 3 are not practical, this is a crucial parameter to consider.

It is obvious that the practical world is full of concrete problems governed by parameters of all kinds that are bounded in small or moderate ranges. If we can design algorithms with running times like $2^k n$ for these problems, then we may have something really useful.

1.4 Kernelization: Another View of FPT

Preprocessing is a practical computing strategy with a lot of power on real world input distributions, as shown by the following example.

Example: Weihe's Train Problem

Weihe describes a problem concerning the train systems of Europe [Wei98]. Consider a bipartite graph $G = (V, E)$ where V is bipartitioned into two sets S (stations) and T (trains), and where an edge represents that a train t stops at a station s. The relevant graphs are huge, on the order of 10,000 vertices. The problem is to compute a minimum number of stations $S' \subseteq S$ such that every train stops at a station in S'. This is a special case of the HITTING SET problem, and is therefore *NP*-complete.

However, the following two reduction rules can be applied to simplify (preprocess) the input to the problem. In describing these rules, let $N(s)$ denote the set of trains that stop at station s, and let $N(t)$ denote the set of stations at which the train t stops.

1. If $N(s) \subseteq N(s')$ then delete s.
2. If $N(t) \subseteq N(t')$ then delete t'.

Applications of these reduction rules cascade, preserving at each step enough information to obtain an optimal solution. Weihe found that, remarkably, these two simple reduction rules were strong enough to "digest" the original, huge input graph into a *problem kernel* consisting of disjoint components of size at most 50 — small enough to allow the problem to be solved optimally by brute force.

The following is an equivalent definition of *FPT* [DFS99].

Definition 5. *A parameterized language L is* kernelizable *if there is there is a parameterized transformation of L to itself, and a function g (unrestricted) that satisfies:*

1. *the running time of the transformation of (x, k) into (x', k'), where $|x| = n$, is bounded by a polynomial $q(n, k)$ (so that in fact this is a polynomial-time transformation of L to itself, considered classically, although with the additional structure of a parameterized reduction),*
2. *$k' \leq k$, and*
3. *$|x'| \leq g(k)$.*

Lemma 1. *A parameterized language L is fixed-parameter tractable if and only if it is kernelizable.*

The proof of this is essentially the solution to the second exercise above.

The kernelization point of view about FPT has become a major enterprise all in itself, that will be covered in the lecture by Saket Saurabh.

There are several points to be noted about kernelization that lead to important research directions:

(1) Kernelization rules are frequently surprising in character, laborious to prove, and nontrivial to discover. Once found, they are small gems of *data reduction* that remain permanently in the heuristic design file for hard problems. No one concerned with any application of HITTING SET on real data should henceforth neglect Weihe's data reduction rules for this problem. The kernelization for VERTEX COVER to graphs of minimum degree 4, for another example, includes the following nontrivial transformation [DFS99]. Suppose G has a vertex x of degree 3 that has three mutually nonadjacent neighbors a, b, c. Then G can be simplified by: (1) deleting x, (2) adding edges from c to all the vertices in $N(a)$, (3) adding edges from a to all the vertices in $N(b)$, (3) adding edges from b to all the vertices in $N(c)$, and (4) adding the edges ab and bc. Note that this transformation is not even symmetric! The resulting (smaller) graph G' has a vertex cover of size k if and only if G has a vertex cover of size k. Moreover, an optimal or good approximate solution for G' lifts constructively to an optimal or good approximate solution for G. The research direction this points to is **to discover these gems of smart preprocessing for all of the hard problems**. There is absolutely nothing to be lost in smart pre-processing, no matter what the subsequent phases of the algorithm (even if the next phase is genetic algorithms or simulated annealing).

(2) Kernelization rules cascade in ways that are surprising, unpredictable in advance, and often quite powerful. Finding a rich set of reduction rules for a hard problem may allow the synergistic cascading of the pre-processing rules to "wrap around" hidden structural aspects of real input distributions. Weihe's train problem provides an excellent example. According to the experience of Alber, Gramm and Niedermeier with implementations of kernelization-based *FPT* algorithms [AGN01], the effort to kernelize is amply rewarded by the subsequently exponentially smaller search tree.

(3) Kernelization is an intrinsically robust algorithmic strategy. Frequently we design algorithms for "pure" combinatorial problems that are not quite like that in practice, because the modeling is only approximate, the inputs are "dirty", etc. For example, what becomes of our VERTEX COVER algorithm if a limited number of edges uv in the graph are *special*, in that it is forbidden to include *both* u and v in the vertex cover? Because they are local in character, the usual kernelization rules are easily adapted to this situation.

(4) Kernelization rules normally preserve all of the information necessary for optimal or approximate solutions. For example, Weihe's kernelization rules for the train problem (HITTING SET) transform the original instance G into a problem kernel G' that can be solved optimally, and the optimal solution for G' "lifts" to an optimal solution for G.

The importance of pre-processing in heuristic design is not a new idea. Cheeseman *et al.* have previously pointed to its importance in the context of artificial intelligence algorithms [CKT91]. What parameterized complexity contributes is a richer theoretical context for this basic element of practical algorithm design. Further research directions include potential methods for mechanizing the discovery and/or verification of reduction rules, and data structures and implementation strategies for efficient kernelization pre-processing.

Lemma 1 of §3 tells us that a parameterized problem is fixed-parameter tractable if and only if there is a polynomial-time kernelization algorithm transforming the input (x, k) into (x', k') where $k' \leq k$ and $|x'| \leq g(k')$ for some function g special to the problem. The basic schema is that reduction rules are applied until an *irreducible* instance (x', k') is obtained. At this point a *Kernel Lemma* is invoked to decide all those reduced instances x' that are larger than $g(k')$ for the kernel-bounding function g. For example, in the cases of Vertex Cover and Planar Dominating Set, if a reduced graph G' is larger than $g(k')$ then (G', k') is a no-instance. In the case of Max Leaf Spanning Tree large reduced instances are automatically yes-instances. (It is notable that for all three of these problems *linear kernelization*, $g(k) = O(k)$, has been established, in all cases nontrivially [CKJ99,FMcRS01,AFN02].)

> **How does one proceed to discover an adequate set of reduction rules, or elucidate (and prove) a bounding function $g(k)$ that insures for instances larger than this bound, that the question can be answered directly?**

We illustrate a systematic approach with the Max Leaf Spanning Tree problem. Our objective is to prove:

The Kernel Lemma. If $(G = (V, E), k)$ is a reduced instance of Max Leaf Spanning Tree and G has more than $g(k)$ vertices, then (G, k) is a yes-instance.

We will prove the Kernel Lemma as a corollary to the following.

The Boundary Lemma. If $G = (V, E)$ is a reduced instance of Max Leaf Spanning Tree that is a yes-instance for k and a no-instance for $k + 1$, then G has at most $h(k)$ vertices.

Let us first verify that the Kernel Lemma follows from the Boundary Lemma. We will make the mild assumption that our functions $g(k)$ and $h(k)$ are nondecreasing. Take $g(k) = h(k)$. Suppose (G, k) is a counterexample to the Kernel Lemma. Then G is reduced, and has more than $h(k)$ vertices, but is a no-instance, that is, G does not have a spanning tree with at least k leaves. Let $k' < k$ be the maximum number of leaves in a spanning tree of G. Then G is a yes-instance for k' and a no-instance for $k' + 1$. Since $k' < k$ and h is non-decreasing, G has more than $h(k')$ vertices, but this contradicts the Boundary Lemma.

The form of the Boundary Lemma (... which still needs to be proved, and we still need to discover what we mean by "reduced", and we also need to identify the particular bounding function h ...) is conducive to an *extremal theorem* style of argument based on a list of inductive priorities. The proof is sketched as follows.

Sketch Proof of the Boundary Lemma. The proof is by *minimum counterexample.* If there is any counterexample, then we can take G to be one that satisfies:
(1) G is reduced.
(2) G is connected and has more than $h(k)$ vertices.
(3) G is a no-instance for $k+1$.
(4) G is a yes-instance for k, as witnessed by an t-rooted tree subgraph T of G that has k leaves. (We do not assume that T is spanning. Note that if T has k leaves then it can be extended to a spanning tree with at least as many leaves.)
(5) G is a counterexample where T has a minimum possible number of vertices.
(6) Among all of the G, T satisfying (1-5), T has a maximum possible number of internal vertices that are adjacent to a leaf of T.
(7) Among all of the G, T satisfying (1-6), the quantity $\sum_{l \in L} d(t,l)$ is minimized, where L is the set of leaves of T and $d(t,l)$ is the distance in T to the "root" vertex t.
Then we argue for a contradiction.

Comment. The point of all this is to set up a framework for argument that will allow us to see what reduction rules are needed, and what $g(k)$ can be achieved. In essence we are setting up a (possibly elaborate, in the spirit of extremal graph theory) argument by minimum counterexample — and using this as a discovery process for the *FPT* algorithm design. The witness structure T of condition (4) gives us a way of "coordinatizing" the situation — giving us some structure to work with in our inductive argument. How this strucuture is used will become clear as we proceed.

We refer to the vertices of $V - T$ as *outsiders.* The following structural claims are easily established. The first five claims are enforced by condition (3), that is, if any of these conditions did not hold, then we could extend T to a tree T' having one more leaf.

Claim 1: No outsider is adjacent to an internal vertex of T.

Claim 2: No leaf of T can be adjacent to two outsiders.

Claim 3: No outsider has three or more outsider neighbors.

Claim 4: No outsider with 2 outsider neighbors is connected to a leaf of T.

Claim 5: The graph induced by the outsider vertices has no cycles.

It follows from Claims (1-5) that the subgraph induced by the outsiders consists of a collection of paths, where the internal vertices of the paths have degree 2 in G. Since we are ultimately attempting to bound the size of G, this suggests (as a discovery process) the following reduction rule for kernelization.

Kernelization Rule 1: If (G, k) has two adjacent vertices u and v of degree 2, then:
(Rule 1.1) If uv is a bridge, then contract uv to obtain G' and let $k' = k$.
(Rule 1.2) If uv is not a bridge, then delete the edge uv to obtain G' and let $k' = k$.

The soundness of this reduction rule is not completely obvious, although not difficult. Having now partly clarified condition (1), we can continue the argument.

The components of the subgraph induced by the outsiders must consist of paths having either 1,2 or 3 vertices.

Because we are trying to efficiently bound the total number of outsiders (as well as everything else, eventually, in order to obtain the best possible kernelization bound $h(k)$), the situation suggests we should look for further reduction rules to address the remaining possible situations with respect to the outsiders. This discovery process leads us to the following further kernelization rules.

Kernelization Rule 2: If (G, k) is a (connected) instance of MAX LEAF where G has a vertex u of degree 1, with neighbor v, and where $\exists x \notin N(v)$ (that is, not every vertex of G is a neighbor of v), then transform (G, k) into (G', k'), where $k = k'$ and G' is obtained by:
(1) deleting u, and
(2) adding edges to make $N[v]$ into a clique.

The reader can verify that this rule is sound: (G, k) is a yes-instance if and only if (G', k') is a yes-instance.

Kernelization Rule 3: If (G, k) is a (connected) instance of MAX LEAF where G has two vertices u and v such that either:
(1) u and v are adjacent, and $N[u] = N[v]$, or
(2) u and v are not adjacent, and $N(u) = N(v)$,
and also (in either case) there is at least one vertex of G not in $N[u] \cup N[v]$, then transform (G, k) to (G', k') where $k' = k - 1$ and G' is obtained by deleting u.

Returning to our consideration of the outsiders, we are now in the situation that for a reduced graph, the only possibilities are:
(1) A component of the outsider graph is a single vertex having at least 2 leaf neighbors in T.
(2) A component of the outsider graph is a K_2 having at least three leaf neighbors in T.
(3) A component of the outsider is a path of three vertices P_3 having at least four leaf neighbors in T.
The weakest of the ratios is given by case (3). We can conclude that the number of outsiders is bounded by $3k/4$.
The next step is to study the tree T. Since it has k leaves, it has at most $k - 2$ branch vertices. Using conditions (5) and (6), but omitting the details, it is argued that: (1) the paths in T between a leaf and its parental branch vertex has no subdivisions, and (2) any other path in T between branch vertices has at most 3 subdivisions (with respect to T). These statements are proved by various further structural claims (as in the analysis of the outsider population) that must hold, else one of the inductive priorities would fail (constructively) — a tree with $k + 1$ leaves would be possible, or a smaller T, or a T with more internal vertices adjacent to leaves can be devised, or one with a better score on the sum-of-distances priority (7). Consequently T has at most $5k$ vertices, unless there is a contradiction. Together with the bound on the outsiders in a reduced graph, this yields a $g(k)$ of $5.75k$. □

The above sketch illustrates how the project of proving an *FPT* kernelization bound is integrated with the search for efficient kernelization rules. But there is more to the story. The argument above also leads directly to a constant-factor polynomial-time approximation algorithm in the following way. First, reduce G using the kernelization rules. It is easy to verify that the rules are approximation-preserving. Thus, we might as well suppose that G is reduced to begin with. Now take *any* tree T (not necessarily spanning) in G. If all of the structural claims hold, then (by our arguments above) the tree T must have at least n/c leaves for $c = 5.75$, and therefore we already have (trivially) a c-approximation. (It would require further arguments, but probably the approximation factor is much better than c.) If at least one of the structural claims does not hold, then the tree T can be improved against one of the inductive priorities. Notice that each claim is proved (in the kernelization argument above) by a constructive consequence. For example, if Claim 1 did not hold, then we can find a tree T' (by modifying T) that has one more leaf. Similarly, each claim violation yields a constructive consequence against one of the inductive priorities in the extremal argument for the kernelization bound. These consequences can be applied to our original T (and its successors) only a polynomial number of times (determined by the list of inductive priorities) until we arrive at a tree T' for which all of the various structural claims hold. At that point, we must have a c-approximate solution.

2 Parameterized Intractability and Structural Complexity

Is there a parameterized analog of Cook's Theorem? Yes there is!

2.1 Various Forms of The Halting Problem: A Central Reference Point

The main investigations of computability and efficient computability are tied to three basic forms of the Halting Problem.

1. THE HALTING PROBLEM
   Input: A Turing machine M.
   Question: If M is started on an empty input tape, will it ever halt?
2. THE POLYNOMIAL-TIME HALTING PROBLEM FOR NONDETERMINISTIC TURING MACHINES
   Input: A nondeterministic Turing machine M.
   Question: Is it possible for M to reach a halting state in n steps, where n is the length of the description of M?
3. THE k-STEP HALTING PROBLEM FOR NONDETERMINISTIC TURING MACHINES
   Input: A nondeterministic Turing machine M and a positive integer k. (The number of transitions that might be made at any step of the computation is unbounded, and the alphabet size is also unrestricted.)
   Parameter: k
   Question: Is it possible for M to reach a halting state in at most k steps?

The first form of the HALTING PROBLEM is useful for studying the question:

"Is there ANY algorithm for my problem?"

The second form of the HALTING PROBLEM has proved useful for nearly 30 years in addressing the question:

"Is there an algorithm for my problem ... like the ones for Sorting and Matrix Multiplication?"

The second form of the HALTING PROBLEM is trivially *NP*-complete, and essentially defines the complexity class *NP*. For a concrete example of why it is trivially *NP*-complete, consider the 3-COLORING problem for graphs, and notice how easily it reduces to the P-TIME NDTM HALTING PROBLEM. Given a graph G for which 3-colorability is to be determined, we just create the following nondeterministic algorithm:

```
Phase 1. (There are $n$ lines of code here if $G$ has $n$ vertices.)
(1.1) Color vertex 1 one of the three colors nondeterministically.
(1.2) Color vertex 2 one of the three colors nondeterministically.
  ...
(1.n) Color vertex $n$ one of the three colors nondeterministically.
Phase 2. Check to see if the coloring is proper and if so halt. Otherwise go into an infinite loop.
```

It is easy to see that the above nondeterministic algorithm has the possibility of halting in m steps (for a suitably padded Turing machine description of size m) if and only if the graph G admits a 3-coloring. Reducing any other problem $\Pi \in NP$ to the P-TIME NDTM HALTING PROBLEM is no more difficult than taking an argument that the problem Π belongs to NP and modifying it slightly to be a reduction to this form of the HALTING PROBLEM. It is in this sense that the P-TIME NDTM HALTING PROBLEM is essentially the *defining* problem for NP.

The conjecture that $P \neq NP$ is intuitively well-founded. The second form of the HALTING PROBLEM would seem to require exponential time because there is little we can do to analyze unstructured nondeterminism other than to exhaustively explore the possible computation paths.

When the question is:

"Is there an algorithm for my problem ... like the one for Vertex Cover?"

the third form of the HALTING PROBLEM anchors the discussion.

The third natural form of the HALTING PROBLEM is trivially solvable in time $O(n^k)$ by exploring the n-branching, depth-k tree of possible computation paths exhaustively. Our intuition here is essentially the same as for the second form of the Halting Problem — that this cannot be improved. The third form of the Halting Problem defines the parameterized complexity class $W[1]$. Thus $W[1]$ is strongly analogous to NP, and the conjecture that $FPT \neq W[1]$ stands on much the same intuitive grounds as the conjecture that $P \neq NP$. The appropriate notion of problem reduction is as follows.

Definition 6. *A parametric transformation from a parameterized language L to a parameterized language L' is an algorithm that computes from input consisting of a pair (x, k), a pair (x', k') such that:*

1. $(x, k) \in L$ *if and only if* $(x', k') \in L'$,
2. $k' = g(k)$ *is a function only of* k, *and*
3. *the computation is accomplished in time* $f(k)n^{\alpha}$, *where* $n = |x|$, α *is a constant independent of both* n *and* k, *and* f *is an arbitrary function.*

Hardness for $W[1]$ is the working criterion that a parameterized problem is unlikely to be *FPT*. The k-Clique problem is $W[1]$-complete [DF98], and often provides a convenient starting point for $W[1]$-hardness demonstrations. This is the parameterized analog of Cook's Theorem, that the third form of the Halting Problem is FPT if and only if the k-Clique problem is FPT.

The main classes of parameterized problems are organized in the tower

$$P \subseteq lin(k) \subseteq poly(k) \subseteq FPT \subseteq M[1] \subseteq W[1] \subseteq M[2] \subseteq W[2] \subseteq \cdots W[P] \subseteq XP$$

2.2 The W[t] Classes

Loosely speaking, the W-hierarchy captures the complexity of the quest for small solutions for constant depth circuits by stepwise increasing the allowed *weft*. The *weft* of a circuit is the maximum number of large gates (of unbounded fan-in) on any input-output path of the circuit. More precisely, W[t] is characterized by the complete problem asking for satisfying assignments of (Hamming-)*weight* k for constant depth circuits of weft t. Here k is the parameter.

Historically, the $W[t]$-hierarchy was inspired by the observation that the parameterized reduction of Clique to the k-weighted satisfiability problem for circuits produces circuits of weft 1 (and depth 2), while the reduction for Dominating Set produces circuits of weft 2, and yet there seems to be no parameterized reduction from Dominating Set to Clique.

Let Γ be a set of circuits. The *k-weighted satisfiability problem of Γ* is the problem WSat(Γ):

Instance: A circuit $\mathcal{C} \in \Gamma$ and a natural k.
Parameter: k.
Problem: Is there an assignment of weight k satisfying $\mathcal{C}$?

Here the *weight* of an assignment is the number of variables that it maps to 1.

W[t] contains all and only the parameterized problems that are for some d fpt reducible to the weighted circuit satisfiability problem WSat($\Omega_{t,d}$) where $\Omega_{t,d}$ is the set of Boolean circuits of weft t and depth at most d. W[P] is defined similarly by WSat(Circ) where Circ is the set of Boolean circuits.

2.3 The M[t] Classes

There is an important class of parameterized problems seemingly intermediate between *FPT* and $W[1]$:

$$FPT \subseteq M[1] \subseteq W[1]$$

There are two natural "routes" to $M[1]$.

The renormalization route to $M[1]$.
There are $O^*(2^{O(k)})$ *FPT* algorithms for many parameterized problems, such as VERTEX COVER. In view of this, we can "renormalize" and define the problem:

$k \log n$ VERTEX COVER
Input: A graph G on n vertices and an integer k; **Parameter:** k; **Question:** Does G have a vertex cover of size at most $k \log n$?

The *FPT* algorithm for the original VERTEX COVER problem, parameterized by the number of vertices in the vertex cover, allows us to place this new problem in XP. It now makes sense to ask whether the $k \log n$ VERTEX COVER problem is also in *FPT* — or is it parametrically intractable? It turns out that $k \log n$ VERTEX COVER is $M[1]$-complete.

The miniaturization route to $M[1]$**.**
We certainly know an algorithm to solve n-variable 3SAT in time $O(2^n)$. Consider the following parameterized problem.

MINI-3SAT
Input: Positive integers k and n in unary, and a 3SAT expression E having at most $k \log n$ variables; **Parameter:** k; **Question:** Is E satisfiable?

Using our exponential time algorithm for 3SAT, MINI-3SAT is in *XP* and we can wonder where it belongs — is it in *FPT* or is it parametrically intractable? This problem also turns out to be complete for $M[1]$.

Dozens of renormalized *FPT* problems and miniaturized arbitrary problems are now known to be $M[1]$-complete. However, what is known is quite problem-specific. For example, one might expect MINI-MAX LEAF to be $M[1]$-complete, but all that is known presently is that it is $M[1]$-hard. It is not known to be $W[1]$-hard, nor is it known to belong to $W[1]$.

The following theorem would be interpreted by most people as indicating that probably $FPT \neq M[1]$. (The theorem is essentially due to Cai and Juedes [CJ01], making use of a result of Impagliazzo, Paturi and Zane [IPZ98].)

Theorem 2. *$FPT = M[1]$ if and only if n-variable 3SAT can be solved in time $2^{o(n)}$.*

$M[1]$ supports convenient although unusual combinatorics. For example, one of the problems that is $M[1]$-complete is the miniature of the INDEPENDENT SET problem defined as follows.

MINI-INDEPENDENT SET
Input: Positive integers k and n in unary, a positive integer $r \leq n$, and a graph G having at most $k \log n$ vertices.

Parameter: k
Question: Does G have an independent set of size at least r?

Theorem 3. *There is an FPT reduction from* MINI-INDEPENDENT SET *to ordinary parameterized* INDEPENDENT SET *(parameterized by the number of vertices in the independent set).*

Proof. Let $G = (V, E)$ be the miniature, for which we wish to determine whether G has an independent set of size r. Here, of course, $|V| \leq k \log n$ and we may regard the vertices of G as organized in k blocks $V_1, ..., V_k$ of size $\log n$. We now employ a simple but useful *counting trick* that can be used when reducing miniatures to "normal" parameterized problems. Our reduction is a Turing reduction, with one branch for each possible way of writing r as a sum of k terms, $r = r_1 + \cdots + r_k$, where each r_i is bounded by $\log n$. The reader can verify that $(\log n)^k$ is an *FPT* function, and thus that there are an allowed number of branches. A branch represents a commitment to choose r_i vertices from block V_i (for each i) to be in the independent set.

We now produce (for a given branch of the Turing reduction) a graph G' that has an independent set of size k if and only if the miniature G has an independent set of size r, distributed as indicated by the commitment made on that branch. The graph G' consists of k cliques, together with some edges between these cliques. The ith clique consists of vertices in 1:1 correspondence with the subsets of V_i of size r_i. An edge connects a vertex x in the ith clique and a vertex y in the jth clique if and only if there is a vertex u in the subset $S_x \subseteq V_i$ represented by x, and a vertex v in the subset $S_y \subseteq V_j$ represented by y, such that $uv \in E$. Verification is straightforward.

The theorem above shows that $M[1]$ is contained in $W[1]$.

Cai and Juedes [CJ01] proved the following, opening up a broad program of studying the optimality of FPT algorithms.

Theorem 4. *If $FPT \neq M[1]$ then there cannot be an FPT algorithm for the general* VERTEX COVER *problem with a parameter function of the form $f(k) = 2^{o(k)}$, and there cannot be an FPT algorithm for the* PLANAR VERTEX COVER *problem with a parameter function of the form $f(k) = 2^{o(\sqrt{k})}$.*

It has previously been known that PLANAR DOMINATING SET, parameterized by the number n of vertices in the graph can be solved optimally in time $O^*(2^{O(\sqrt{n})})$ by using the Lipton-Tarjan Planar Separator Theorem. Combining the lower bound theorem of Cai-Juedes with the linear kernelization result of Alber et al. [AFN02] shows that this cannot be improved to $O^*(2^{o(\sqrt{n})})$ unless $FPT = M[1]$.

2.4 An Example of a $W[1]$-hardness Reduction

We take as our example, how parameterized complexity can be used to study the complexity of approximation. Approximation immediately concerns a fundamental parameter: $k = 1/\epsilon$, *the goodness of the approximation.*

To illustrate the issue, consider the following more-or-less random sample of recent PTAS results:

- The PTAS for the Euclidean TSP due to Arora [Ar96] has a running time of around $O(n^{3000/\epsilon})$. Thus for a 20% error, we have a "polynomial-time" algorithm that runs in time $O(n^{15000})$.
- The PTAS for the Multiple Knapsack problem due to Chekuri and Khanna [CK00] has a running time of $O(n^{12(\log(1/\epsilon)/\epsilon^8)})$. Thus for a 20% error we have a polynomial-time algorithm that runs in time $O(n^{9375000})$.
- The PTAS for the Minimum Cost Routing Spanning Tree problem due to Wu, Lancia, Banfna, Chao, Ravi and Tang [WLBCRT98] has a running time of $O(n^{2\lceil 2/\epsilon\rceil-2})$. For a 20% error, we thus have a running time of $O(n^{18})$.
- The PTAS for the Unbounded Batch Scheduling problem due to Deng, Feng, Zhang and Zhu [DFZZ01] has a running time of $O(n^{5\log_{1+\epsilon}(1+(1/\epsilon))})$. Thus for a 20% error we have an $O(n^{50})$ polynomial-time algorithm.
- The PTAS for Two-Vehicle Scheduling on a Path due to Karuno and Nagamochi [KN01] has a running time of $O(n^{8(1+(2/\epsilon))})$; thus $O(n^{88})$ for a 20% error.
- The PTAS for the Maximum Subforest Problem due to Shamir and Tsur [ST98] has a running time of $O(n^{2^{2^{1/\epsilon}}-1})$. For a 20% error we thus have a "polynomial" running time of $O(n^{958267391})$.
- The PTAS for the Maximum Indendent Set problem on geometric graphs due to Erlebach, Jansen and Seidel [EJS01] has a running time of $O(n^{(4/\pi)(1/\epsilon^2+2)^2(1/\epsilon^2+1)^2})$. Thus for a 20% error we have a running time of $O(n^{532804})$.
- The PTAS for the Class-Constrained Packing Problem due to Shachnai and Tamir [ST00] has a running time (for 3 colors) of $O(n^{64/\epsilon+(log(1/\epsilon)/\epsilon^8)})$. Thus for a 20% error (for 3 colors) we have a running time of $O(n^{1021570})$.
- The PTAS for the problem of Base Station Positioning in UMTS Networks due to Galota, Glasser, Reith and Vollmer [GGRV01] has a running time of $O(n^{25/\epsilon^2})$, and thus $O(n^{627})$ time for a 20% error.
- The PTAS for the General Multiprocessor Job Scheduling Problem due to Chen and Miranda [CM99] runs in time $O(n^{(3mm!)^{(m/\epsilon)+1}})$ for m machines. Thus for 4 machines with a 20% error we have an algorithm that runs in time $O(n^{1000000000000000000000000000000000000000000000000000000000000000000000})$ or so.

Since polynomial-time algorithms with exponent greater than 3 are generally not very practical, the following question would seem to be important.

Can we get the $k = 1/\epsilon$ out of the exponent?

The following definition captures the essential issue.

Definition 7. *An optimization problem Π has an* efficient P-time approximation scheme *(EPTAS) if it can be approximated to a goodness of* $(1+\epsilon)$ *of optimal in time $f(k)n^c$ where c is a constant and $k = 1/\epsilon$.*

In 1997, Arora gave an EPTAS for the Euclidean TSP [Ar97].

The following easy but important connection between parameterized complexity and approximation was first proved by Bazgan [Baz95,CT97].

Theorem 5. *Suppose that Π_{opt} is an optimization problem, and that Π_{param} is the corresponding parameterized problem, where the parameter is the value of an optimal solution. Then Π_{param} is fixed-parameter tractable if Π_{opt} has an efficient PTAS.*

Applying Bazgan's Theorem is not necessarily difficult — we will sketch here a recent example. Khanna and Motwani introduced three planar logic problems in an interesting effort to give a general explanation of PTAS-approximability. Their suggestion is that "hidden planar structure" in the logic of an optimization problem is what allows PTASs to be developed [KM96]. They gave examples of optimization problems known to have PTASs, problems having nothing to do with graphs, that could nevertheless be reduced to these planar logic problems. The PTASs for the planar logic problems thus "explain" the PTASs for these other problems. Here is one of their three general planar logic optimization problems.

Planar TMIN
Input: A collection of Boolean formulas in sum-of-products form, with all literals positive, where the associated bipartite graph is planar (this graph has a vertex for each formula and a vertex for each variable, and an edge between two such vertices if the variable occurs in the formula).
Output: A truth assignment of minimum weight (i.e., a minimum number of variables set to *true*) that satisfies all the formulas.

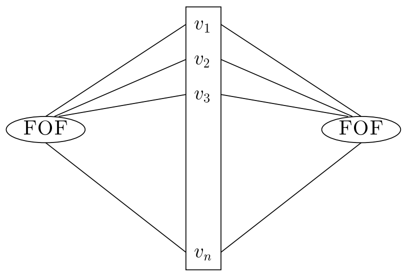
The following theorem is from joint work with Cai, Juedes and Rosamond [CFJR01].

Theorem 6. *Planar TMIN is hard for $W[1]$ and therefore does not have an EPTAS unless $FPT = W[1]$.*

Proof. We show that Clique is parameterized reducible to Planar TMIN with the parameter being the weight of a truth assignment. Since Clique is W[1]-complete, it will follow that the parameterized form of Planar TMIN is W[1]-hard.

To begin, let $\langle G, k\rangle$ be an instance of Clique. Assume that G has n vertices. From G and k, we will construct a collection C of FOFs (sum-of-products formulas) over $f(k)$ blocks of n variables. C will contain at most $2f(k)$ FOFs and the incidence graph of C will be planar. Moreover, each minterm in each FOF will contain at most 4 variables. The collection C is constructed so that G has a clique of size k if and only if C has a weight $f(k)$ satisfying assignment with exactly one variable set to true in each block of n variables. Here we have that $f(k) = O(k^4)$.

To maintain planarity in the incidence graph for C, we ensure that each block of n variables appears in at most 2 FOFs. If this condition is maintained, then we can draw each block of n variables as follows.

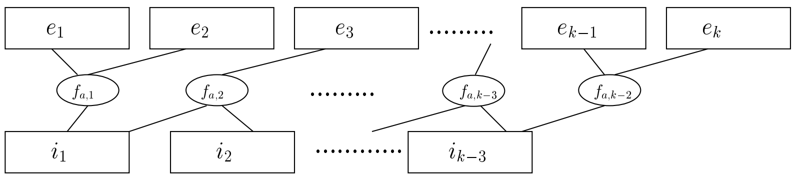


We describe the construction in two stages. In the first stage, we use k blocks of n variables and a collection C' of $k(k-1)/2+k$ FOFs. In a weight k satisfying assignment for C', exactly one variable v_i, j in each block of variables $b_i = [v_{i,1}, \ldots, v_{i,n}]$ will be set to true. We interpret this event as "vertex j is the ith vertex in the clique of size k." The $k(k-1)/2+k$ FOFs are described as follows. For each $1 \leq i \leq k$, let f_i be the FOF $\bigvee\limits_{j=1}^{n} v_{i,j}$. This FOF ensures that at least one variable in b_i is set to true. For each pair $1 \leq i < j \leq k$, let $f_{i,j}$ be the FOF $\bigvee\limits_{(u,v)\in E} v_{i,u}v_{j,v}$. Each FOF $f_{i,j}$ ensures that there is an edge in G between the ith vertex the clique and the jth vertex in the clique.

It is somewhat straightforward to show that $C' = \{f_1, \ldots, f_k, f_{1,2}, \ldots, f_{k-1,k}\}$ has a weight k satisfying assignment if and only if G has a clique of size k. To see this, notice that any weight k satisfying assignment for C' must satisfy exactly 1 variable in each block b_i. Each first order formula $f_{i,j}$ ensures that there is an edge between the ith vertex in the potential clique and the jth vertex in the potential clique. Notice also that, since we assume that G does not contain edges of the form (u, u), the FOF $f_{i,j}$ also ensures that the ith vertex in the potential clique is not the jth vertex in the potential clique. This completes the first stage.

The incidence graph for the collection C' in the first stage is almost certainly not planar. In the second stage, we achieve planarity by removing crossovers in incidence graph for C'. Here we use two types of widgets to remove crossovers while keeping the number of variables per minterm bounded by 4. The first widget A_k consists of $k + k - 3$ blocks of n variables and $k - 2$ FOFs. This widget consists of $k-3$ internal and k external blocks of variables. Each external block $e_i = [e_{i,1}, \ldots, e_{i,n}]$ of variables is connected to exactly one FOF inside the widget. Each internal block $i_j = [i_{j,1}, \ldots, e_{j,n}]$ is connected to exactly two FOFs inside the widget. The $k - 2$ FOFs are given as follows. The FOF $f_{a,1}$ is $\bigvee\limits_{j=1}^{n} e_{1,j}e_{2,j}i_{1,j}$. For each $2 \leq l \leq k - 3$, the FOF $f_{a,l} = \bigvee_{j=1}^{n} i_{l-1,j}e_{l+1,j}i_{l,j}$. Finally, $f_{a,k-2} = \bigvee\limits_{j=1}^{n} i_{k-3,j}e_{k-1,j}e_{k,j}$. These $k-2$ FOFs ensure that the settings of variables in each block is the same if there is a weight $2k - 3$ satisfying assignment to the $2k - 3$ blocks of n variables.

The widget A_k can be drawn as follows.

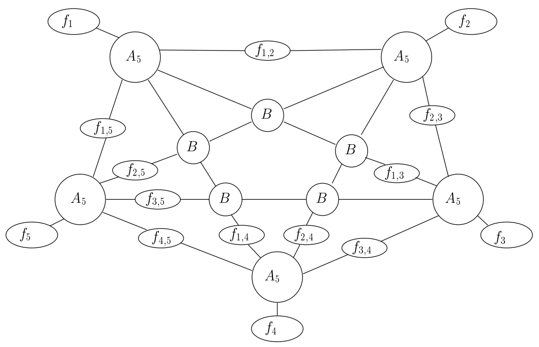


Since each internal block is connected to exactly two FOFs, the incidence graph for this widget can be drawn on the plane without crossing any edges.

The second widget removes crossover edges from the first stage of the construction. In the first stage, crossovers can occur in the incidence graphs because two FOFs may cross from one block to another. To eliminate this, consider each edge i, j in K_k with $i < j$ as a directed edge from i to j. In the construction, we send a copy of block i to block j. At each crossover point from the direction of block $u = [u_1, \ldots, u_n]$ and $v = [v_1, \ldots, v_n]$, insert a widget B that introduces 2 new blocks of n variables $u_1 = [u_{1_1} \ldots u_{1_n}]$ and $v_1 = [v_{1_1} \ldots v_{1_n}]$ and a FOF $f_B = \bigvee_{j=1}^{n} \bigvee_{l=1}^{n} u_j u_{1_j} v_l v_{1_l}$. The FOF f_B ensures that u_1 and v_1 are copies of u and v. Moreover, notice that the incidence graph for the widget B is also planar.

To complete the construction, we replace each of the original k blocks of n variables from the first stage with a copy of the widget A_{k-1}. At each crossover point in the graph, we introduce a copy of widget B. Finally, for each directed edge between blocks (i, j), we insert the original FOF $f_{i,j}$ between the last widget B and the destination widget A_{k-1}. Since one of the new blocks of variables created by the widget B is a copy of block i, the effect of the FOF $f_{i,j}$ in this new collections is the same as before.

The following diagram shows the full construction when $k = 5$.



Since each the incidence graph of each widget in this drawing is planar, the entire collection C of first order formulas has a planar incidence graph.

Now, if we assume that there are $c(k) = O(k^4)$ crossover points in standard drawing of K_k, then our collection has $c(k)$ B widgets. Since each B widget introduces 2 new blocks of n variables, this gives $2c(k)$ new blocks. Since we have k A_{k-1} widgets, each of which has $2(k-1) - 3 = 2k - 5$ blocks of n variables, this gives an additional $k(2k-5)$ blocks. So, in total, our construction has $f(k) = 2c(k) + 2k^2 - 5k = O(k^4)$ blocks of n variables. Note also that there are $g(k) = k(k-1)/2 + k(k-2) + c(k) = O(k^4)$ FOFs in the collection C.

As shown in our construction C has a weight $f(k)$ satisfying assignment (i.e., each block has exactly one variable set to true) if and only if the original graph G has a clique of size k. Since the incidence graph of C is planar and each minterm in each FOF contains at most four variables, it follows that this construction is a parameterized reduction as claimed. $\square$

In a similar manner the other two planar logic problems defined by Khanna and Motwani can also be shown to be $W[1]$-hard.

3 Recommended Books and Articles

Parameterized Complexity – R. Downey and M. Fellows, Springer, 1999.

Parameterized Complexity Theory – J. Flum and M. Grohe, Springer, 2006.

Invitation to Fixed Parameter Algorithms – R. Niedermeier, Oxford Univ. Press, 2006.

The Computer Journal, 2008, Numbers 1 and 3 – a double special issue of surveys of various aspects and application areas of parameterized complexity.

References

[ADF95] K. Abrahamson, R. Downey and M. Fellows, "Fixed Parameter Tractability and Completeness IV: On Completeness for $W[P]$ and *PSPACE* Analogs," *Annals of Pure and Applied Logic* 73 (1995), 235–276.

[AFN01] J. Alber, H. Fernau and R. Niedermeier, "Parameterized Complexity: Exponential Speed-Up for Planar Graph Problems," in: Proceedings of ICALP 2001, Crete, Greece, *Lecture Notes in Computer Science* vol. 2076 (2001), 261-272.

[AFN02] J. Alber, M. Fellows and R. Niedermeier, "Efficient Data Reduction for Dominating Set: A Linear Problem Kernel for the Planar Case," to appear in the *Proceedings of Scandinavian Workshop on Algorithms and Theory* (SWAT 2002), Springer-Verlag, *Lecture Notes in Computer Science*, 2002.

[AGN01] J. Alber, J. Gramm and R. Niedermeier, "Faster Exact Algorithms for Hard Problems: A Parameterized Point of View," *Discrete Mathematics* 229 (2001), 3–27.

[Ar96] S. Arora, "Polynomial Time Approximation Schemes for Euclidean TSP and Other Geometric Problems," In: *Proceedings of the 37th IEEE Symposium on Foundations of Computer Science*, 1996, pp. 2–12.

[Ar97] S. Arora, "Nearly Linear Time Approximation Schemes for Euclidean TSP and Other Geometric Problems," *Proc. 38th Annual IEEE Symposium on the Foundations of Computing* (FOCS'97), IEEE Press (1997), 554-563.

[Baz95] C. Bazgan, "Schémas d'approximation et complexité paramétrée," Rapport de stage de DEA d'Informatique à Orsay, 1995.

[BR99] N. Bansal and V. Raman, "Upper Bounds for MaxSat: Further Improved," *Proc. 10th International Symposium on Algorithms and Computation (ISAAC '99)*, Springer-Verlag, *Lecture Notes in Computer Science* 1741 (1999), 247–258.

[CFJR01] Liming Cai, M. Fellows, D. Juedes and F. Rosamond, "Efficient Polynomial-Time Approximation Schemes for Problems on Planar Structures: Upper and Lower Bounds," manuscript, 2001.

[CJ01] L. Cai and D. Juedes. "On the Existence of Subexponential Parameterized Algorithms," manuscript, 2001. Revised version of the paper, "Subexponential Parameterized Algorithms Collapse the W-Hierarchy," in: *Proceedings 28th ICALP*, Springer-Verlag LNCS 2076 (2001), 273–284.

[CK00] C. Chekuri and S. Khanna, "A PTAS for the Multiple Knapsack Problem," *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms* (SODA 2000), pp. 213-222.

[CKJ99] J. Chen, I.A. Kanj and W. Jia, "Vertex Cover: Further Observations and Further Improvements," *Proceedings of the 25th International Workshop on Graph-Theoretic Concepts in Computer Science* (WG'99), *Lecture Notes in Computer Science 1665* (1999), 313–324.

[CKT91] P. Cheeseman, B. Kanefsky and W. Taylor, "Where the Really Hard Problems Are," *Proc. 12th International Joint Conference on Artificial Intelligence* (1991), 331-337.

[CM99] J. Chen and A. Miranda, "A Polynomial-Time Approximation Scheme for General Multiprocessor Scheduling," *Proc. ACM Symposium on Theory of Computing* (STOC '99), ACM Press (1999), 418–427.

[CS97] Leizhen Cai and B. Schieber, "A Linear Time Algorithm for Computing the Intersection of All Odd Cycles in a Graph," *Discrete Applied Math.* 73 (1997), 27-34.

[CT97] M. Cesati and L. Trevisan, "On the Efficiency of Polynomial Time Approximation Schemes," *Information Processing Letters* 64 (1997), 165–171.

[CW95] M. Cesati and H. T. Wareham, "Parameterized Complexity Analysis in Robot Motion Planning," *Proceedings 25th IEEE Intl. Conf. on Systems, Man and Cybernetics: Volume 1*, IEEE Press, Los Alamitos, CA (1995), 880-885.

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

[DFS99] R. G. Downey, M. R. Fellows and U. Stege, "Parameterized Complexity: A Framework for Systematically Confronting Computational Intractability." In: *Contemporary Trends in Discrete Mathematics*, (R. Graham, J. Kratochvíl, J. Nesetril and F. Roberts, eds.), Proceedings of the DIMACS-DIMATIA Workshop, Prague, 1997, *AMS-DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol. 49 (1999), 49–99.

[DFT96] R. G. Downey, M. Fellows and U. Taylor, "The Parameterized Complexity of Relational Database Queries and an Improved Characterization of $W[1]$," in: *Combinatorics, Complexity and Logic: Proceedings of DMTCS'96*, Springer-Verlag (1997), 194–213.

[DFZZ01] X. Deng, H. Feng, P. Zhang and H. Zhu, "A Polynomial Time Approximation Scheme for Minimizing Total Completion Time of Unbounded Batch Scheduling," *Proc. 12th International Symposium on Algorithms and Computation* (ISAAC '01), Springer-Verlag, *Lecture Notes in Computer Science* 2223 (2001), 26–35.

[EJS01] T. Erlebach, K. Jansen and E. Seidel, "Polynomial Time Approximation Schemes for Geometric Graphs," *Proc. ACM Symposium on Discrete Algorithms* (SODA'01), 2001, pp. 671–679.

[FG01] J. Flum and M. Grohe, "Describing Parameterized Complexity Classes," manuscript, 2001.

[FMcRS01] M. Fellows, C. McCartin, F. Rosamond and U. Stege, "Trees with Few and Many Leaves," manuscript, full version of the paper: "Coordinatized kernels and catalytic reductions: An improved FPT algorithm for max leaf spanning tree and other problems," *Proceedings of the 20th FST TCS Conference*, New Delhi, India, Lecture Notes in Computer Science vol. 1974, Springer Verlag (2000), 240-251.

[GGRV01] M. Galota, C. Glasser, S. Reith and H. Vollmer, "A Polynomial Time Approximation Scheme for Base Station Positioning in UMTS Networks," *Proc. Discrete Algorithms and Methods for Mobile Computing and Communication*, 2001.

[GN00] J. Gramm and R. Niedermeier, "Faster Exact Algorithms for Max2Sat," *Proc. 4th Italian Conference on Algorithms and Complexity*, Springer-Verlag, *Lecture Notes in Computer Science* 1767 (2000), 174–186.

[Gr01a] M. Grohe, "Generalized Model-Checking Problems for First-Order Logic," *Proc. STACS 2001*, Springer-Verlag LNCS vol. 2001 (2001), 12–26.

[Gr01b] M. Grohe, "The Parameterized Complexity of Database Queries," *Proc. PODS 2001*, ACM Press (2001), 82–92.

[GSS01] G. Gottlob, F. Scarcello and M. Sideri, "Fixed Parameter Complexity in AI and Nonmonotonic Reasoning," to appear in *The Artificial Intelligence Journal.* Conference version in: *Proc. of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning* (LPNMR'99), vol. 1730 of *Lecture Notes in Artificial Intelligence* (1999), 1–18.

[HM91] F. Henglein and H. G. Mairson, "The Complexity of Type Inference for Higher-Order Typed Lambda Calculi." In *Proc. Symp. on Principles of Programming Languages (POPL)* (1991), 119-130.

[IPZ98] R. Impagliazzo, R. Paturi and F. Zane. "Which Problems Have Strongly Exponential Complexity?" *Proceedings of the 39th Annual Symposium on Foundations of Computer Science (FOCS'1998)*, 653–663.

[KM96] S. Khanna and R. Motwani, "Towards a Syntactic Characterization of PTAS," in: *Proc. STOC 1996*, ACM Press (1996), 329–337.

[KN01] Y. Karuno and H. Nagamochi, "A Polynomial Time Approximation Scheme for the Multi-vehicle Scheduling Problem on a Path with Release and Handling Times," *Proc. 12th International Symposium on Algorithms and Computation* (ISAAC '01), Springer-Verlag, *Lecture Notes in Computer Science* 2223 (2001), 36–47.

[KR00] S. Khot and V. Raman, 'Parameterized Complexity of Finding Subgraphs with Hereditary properties', *Proceedings of the Sixth Annual International Computing and Combinatorics Conference (COCOON 2000)* July 2000, Sydney, Australia, Lecture Notes in Computer Science, Springer Verlag **1858** (2000) 137-147.

[LP85] O. Lichtenstein and A. Pneuli. "Checking That Finite-State Concurrents Programs Satisfy Their Linear Specification." In: *Proceedings of the 12th ACM Symposium on Principles of Programming Languages* (1985), 97–107.

[MR99] M. Mahajan and V. Raman, "Parameterizing Above Guaranteed Values: MaxSat and MaxCut," *J. Algorithms* 31 (1999), 335-354.

[NR01] R. Niedermeier and P. Rossmanith, "An Efficient Fixed Parameter Algorithm for 3-Hitting Set," *Journal of Discrete Algorithms* 2(1), 2001.

[NSS98] A. Natanzon, R. Shamir and R. Sharan, "A Polynomial-Time Approximation Algorithm for Minimum Fill-In," *Proc. ACM Symposium on the Theory of Computing* (STOC'98), ACM Press (1998), 41–47.

[PY97] C. Papadimitriou and M. Yannakakis, "On the Complexity of Database Queries," *Proc. ACM Symp. on Principles of Database Systems* (1997), 12–19.

[ST98] R. Shamir and D. Tzur, "The Maximum Subforest Problem: Approximation and Exact Algorithms," *Proc. ACM Symposium on Discrete Algorithms* (SODA'98), ACM Press (1998), 394–399.

[ST00] H. Shachnai and T. Tamir, "Polynomial Time Approximation Schemes for Class-Constrained Packing Problems," *Proc. APPROX 2000.*

[St00] U. Stege, "Resolving Conflicts in Problems in Computational Biochemistry," Ph.D. dissertation, ETH, 2000.

[Wei98] K. Weihe, "Covering Trains by Stations, or the Power of Data Reduction," *Proc. ALEX'98* (1998), 1–8.

[Wei00] K. Weihe, "On the Differences Between 'Practical' and 'Applied' (invited paper)," *Proc. WAE 2000*, Springer-Verlag, *Lecture Notes in Computer Science* 1982 (2001), 1–10.

[WLBCRT98] B. Wu, G. Lancia, V. Bafna, K-M. Chao, R. Ravi and C. Tang, "A Polynomial Time Approximation Scheme for Minimum Routing Cost Spanning Trees," *Proc. SODA '98*, 1998, pp. 21–32.