

3. Parameterized Complexity: The Main Ideas and Connections to Practical Computing

Michael R. Fellows

School of Electrical Engineering and Computer Science
University of Newcastle, University Drive, Callaghan NSW 2308, Australia
mfellows@cs.newcastle.edu.au

Summary.

The purposes of this paper are two:

- (1) to give an exposition of the main ideas of parameterized complexity, and
- (2) to discuss the connections of parameterized complexity to the systematic design of heuristics and approximation algorithms.

3.1 Introduction

Research in the parameterized framework of complexity analysis, and on the corresponding toolkit of algorithm design methods has been expanding rapidly in recent years. This has led to a flurry of recent surveys, all of which are good sources of introductory material [3.46, 3.42, 3.22, 3.24, 3.3, 3.32, 3.33]. One could also turn to the monograph [3.21]. Experience with implementations of *FPT* algorithms is described in [3.34, 3.49, 3.3]. In several cases, these implementations now provide the best available “heuristic” algorithms for general well-known *NP*-hard problems. More importantly, the theory seems to provide some useful mathematical systematization of much existing practice in heuristics and practical computing. Computing practitioners have naturally exploited limited structural parameter ranges of the problem inputs they have been faced with, or limited parameter ranges of the solutions they have sought.

The first part of this survey summarizes the main ideas of parameterized complexity and presents a broad perspective on the program. The second part is concerned with connections to heuristics and practical computing strategies, and to approximation algorithms. Thus Section 3.2 gives the overview and main definitions. Section 3.3 explores the natural relationship of fixed-parameter tractability to the design of practical algorithms and gives several examples showing the apparently widespread situation that many industrial strength heuristics currently in use are in fact *FPT* algorithms for natural parameters, previously uncharted as such. Section 3.4 describes how parameterization according to the goodness of approximation ($k = 1/\epsilon$ for approximations to within a factor of $1 + \epsilon$ of optimal) provides a vital critical tool for evaluating the practical significance of recent work on polynomial-time approximation schemes. Section 3.5 explores how *FPT* algorithm design is

naturally intertwined with polynomial-time approximation algorithms and pre-processing based heuristics.

3.2 Parameterized Complexity in a Nutshell

The main ideas of parameterized complexity are organized here into two discussions:

- The basic empirical motivation.
- The perspective provided by forms of the Halting Problem.

3.2.1 Empirical Motivation: Two Forms of Fixed-Parameter Complexity

Most 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. After many rounds of improvement involving a variety of ideas, starting from a simple $O(2^k n)$ algorithm, the best known algorithm for VERTEX COVER now runs in time $O(1.271^k + kn)$ [3.17]. This is implemented and practical for n of unlimited size and k up to around 400 [3.34, 3.49, 3.19].
2. The best known algorithm for DOMINATING SET is *still* just the brute force algorithm of trying all k -subsets. For a graph on n vertices this approach has a running time of $O(n^{k+1})$.

Table 3.2.1 shows the contrast between these two kinds of complexity.

In order to formalize the difference between VERTEX COVER and DOMINATING SET we make the following basic definitions.

Definition 3.2.1. 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.

Table 3.1. The ratio $\frac{n^{k+1}}{2^k n}$ for various values of n and k

	$n = 50$	$n = 100$	$n = 150$
$k = 2$	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}

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

Definition 3.2.2. 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 3.2.3. 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).

As an exercise, the reader might wish to 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.

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* [3.23, 3.44] because it is hard for $W[1]$ (a form of negative evidence explained in Section 3.2.2). However, if the parameter is the size of the query and the treewidth of the database, then the problem is fixed-parameter tractable. It appears that many databases encountered in practice do have bounded treewidth, so this quite nontrivial *FPT* result has significant potential to be useful [3.33].
- *The nesting depth of a logical expression.* ML compilers work reasonably well. One of the problems the compiler must solve is the checking of the compatibility of type declarations. This problem is complete for deterministic exponential time [3.35], so the situation appears dire from the standpoint of complexity theory. The implementations work well in practice, using an algorithm that previously would have been called a heuristic because — we can now say — the ML TYPE CHECKING problem is solved by an *FPT* algorithm 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 [3.39]. Since normally $k \leq 6$, the algorithm is clearly practical.

- *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 [3.24].

- *The number of processors in a practical parallel processing system.* This is frequently in the range of $k \leq 64$. Is there a practical and interesting theory of parallel *FPT*? For a recent paper that explores practical parallel implementations of *FPT* algorithms see [3.19]. (Coarse-grained parallel implementations of *FPT* algorithms for VERTEX COVER are accessible on an algorithmic server at the University of Carleton website.)

- *The number of variables or clauses in a logical formula, or the number of steps in a deductive procedure.* Some initial studies of applications of parameterized complexity to logic programming and artificial intelligence have recently appeared [3.50, 3.30]. Much remains unexplored. Determining whether at least k clauses of a CNF formula F are satisfiable is *FPT* with a running time of $O(|F| + 1.381^k k^2)$ [3.8]. 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 $O(|F| + 6.92^k k^2)$ [3.8]. Implementations indicate that these algorithms are quite practical [3.31], with the kernelization transformations of the *FPT* algorithms having particular practical value in decreasing the size of the subsequent exponential search trees.

- *The number of steps for a motion planning problem.* Where the description of the terrain has size n (which therefore bounds the number of movement options at each step), we can solve this problem in time $O(n^{k+1})$ trivially. Are there significant classes of motion planning problems that are fixed-parameter tractable? Exploration of this topic has hardly begun [3.14].

- *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 [3.1]. 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. This is potentially a very fruitful area, since games are used to model many different kinds of situations.

- *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. This does not appear to be practical on the basis of this parameter function $f(k)$, but the algorithm has been implemented and appears to be practical for k up to 500 for classes of randomly generated tests. This seems to be the best available algorithm for PLANAR DOMINATING SET.

- *A “dual” parameter.* A graph has an independent set of size k if and only if it has a vertex cover of size $n - k$. Many problems have such a natural dual form and it is “almost” a general rule, first noted by Raman, that parametric duals of *NP*-hard problems have complementary parameterized complexity (one is *FPT*, and the other is *W*[1]-hard) [3.38, 3.6]. For example, $n - k$ DOMINATING SET is *FPT*, as is $n - k$ GRAPH COLORING. Solving a hard problem by parameterizing from “the other end” appears to be an important and general algorithmic strategy. The best available algorithm for MAXIMUM INDEPENDENT SET is to compute a VERTEX COVER by the very good *FPT* algorithms for this problem, and take the complement.

- *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. Problems of this sort have recently begun to receive attention [3.11].

- *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 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 [3.43].

- *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 .Is it *FPT* to explore the k -change neighborhood for TSP?

• *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. The reader will find more about this in Section 3.4.

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.

The following definition provides us with a place to put all those problems that are “solvable in polynomial time for fixed k ” without making our central distinction about whether this “fixed k ” is ending up in the exponent or not.

Definition 3.2.4. *A parameterized language L belongs to the class XP (slice-wise P) 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 [3.21].

Theorem 3.2.1. *FPT is a proper subset of XP .*

3.2.2 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. Apart from accumulated habit, this concrete intuition is the fundamental reference point for classical complexity theory.

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. This question will increasingly and inevitably be asked for any NP -hard problem for which small parameter ranges of input or output aspects or structure are important in applications. It is reasonable to assert that there are few applications of computing where this will not be true.

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 3.2.5. *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 [3.21], and often provides a convenient starting point for $W[1]$ -hardness demonstrations.

3.3 Connections to Practical Computing and Heuristics

What is the working context of practical computing? A thought-provoking account of this subject has been given by Weihe [3.52].

A crucial question is: *What are the actual inputs that practical computing implementations have to deal with?*

In considering “war stories” of practical computing, such as reported by Weihe, we are quickly forced to give up the idea that real inputs (for most problems) fill up the definitional spaces of our mathematical modeling. The general rule also is that real inputs are *not random*, but rather have lots of hidden structure that may not have a familiar name or conceptualization.

Example 1: Weihe’s Train Problem

Weihe describes a problem concerning the train systems of Europe [3.51]. 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' . It is easy to see that this is a special case of the HITTING SET problem, and is therefore NP -complete. Moreover, it is also $W[1]$ -hard [3.21], so the straightforward application of the parameterized complexity program seems to fail as well.

However, the following two reduction rules can be applied to simplify (pre-process) 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.

Note that in the same breath, we have here a polynomial-time constant factor approximation algorithm, getting us a solution within a factor of 50 of optimal in, say, $O(n^2)$ time, just by taking *all* the vertices in the kernel components.

Weihe’s example displays a universally applicable coping strategy for hard problems: *smart pre-processing*. It would be silly *not* to undertake pre-processing for an NP -hard problem, even if the next phase is simulated annealing, neural nets, roaming ants, genetic, memetic or the kitchen sink. In a precise sense, this is *exactly* what fixed-parameter tractability is all about. The following is an equivalent definition of FPT [3.24].

Definition 3.3.1. *A parameterized language L is kernelizable if there is a parametric transformation of L to itself, and a function h (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 parametric reduction),*
2. *$k' \leq k$, and*
3. *$|x'| \leq h(k)$, where h is an arbitrary function.*

Lemma 3.3.1. *A parameterized language L is fixed-parameter tractable if and only if it is kernelizable.*

Weihe’s example looks like an FPT kernelization, but what is the parameter? As a thought experiment, let us define $K(G)$ for a bipartite graph G to be the maximum size of a component of G when G is reduced according to the two simple reduction rules above. Then it is clear, although it might seem

artificial, that HITTING SET can be solved optimally in FPT time for the parameter $K(G)$. We can add this new tractable parameterization of HITTING SET to the already known fact that HITTING SET can be solved optimally in FPT -time for the parameter *treewidth*. (It is not hard to show that *treewidth* and $K(G)$ are unrelated.)

As an illustration of the power of pre-processing, the reader will easily discover a reduction rule for VERTEX COVER that eliminates all vertices of degree 1. Not so easy is to show that all vertices of degree at most 3 can be eliminated, leaving as a kernel a graph of minimum degree four. This pre-processing routine yields the best known heuristic algorithm for the general VERTEX COVER problem (i.e., no assumption that k is small), and also plays a central role in the best known FPT algorithm for VERTEX COVER [3.17].

We see in Weihe’s train problem an example of a problem where the natural input distribution (graphs of train systems) occupies a limited parameter range, but the relevant parameter is not at all obvious. The inputs to one computational process (e.g., Weihe’s train problem) are often the outputs of another process (the building and operating of train systems) that also are governed by computational and other feasibility constraints. We might reasonably adopt the view that the real world of computing involves a vast commerce in hidden structural parameters.

Weihe’s algorithm is a beautiful example of an FPT algorithm that exploits a significant hidden structural parameter of the graphs that arise in analyzing train systems, and follows a “classic” pattern in FPT algorithm design: (1) a P -time kernelization, pre-processing phase, followed by (2) an exponential search phase (exponential in the size of the kernel).

Example 2: Heuristics for Breakpoint Phylogeny

It is assumed that there is a fixed set \mathcal{G} of genes shared by all the species for which an evolutionary tree is to be constructed. With each species S is associated a circular ordering of \mathcal{G} where each gene occurs signed, either positively or negatively, according to the transcription direction for the gene (and each gene occurs in the circular ordering for the species exactly once). Given two such circular orderings for species S and S' , a *breakpoint* is an ordered pair of genes (g, g') such that g and g' occur consecutively in S in that order, but neither (g, g') nor $(-g', -g)$ occur consecutively in that order in S' . The *breakpoint distance* $d(S, S')$ between S and S' is the number of breakpoints between S and S' . The *breakpoint score* of a tree labeled at each node with a signed circular ordering of \mathcal{G} is the sum of the breakpoint distances along the edges of the tree.

The BREAKPOINT PHYLOGENY problem takes as input a set of signed circular orderings S_1, \dots, S_n and seeks to find a tree T with n leaves such that:

1. The leaves are labeled 1:1 with the S_i .
2. The internal vertices are labeled with signed circular orderings of \mathcal{G} that represent hypothesized ancestral species.
3. The breakpoint score of T is minimized.

The problem is apparently quite difficult; it is *NP*-complete, even for the seemingly severe restriction to $n = 3$ — only three species! — known as the BREAKPOINT MEDIAN PROBLEM [3.45]. A substantial speedup is reported by Moret, et al. [3.40], for a heuristic based on the following kernelization rule.

- If two genes *always* occur consecutively (either as (g, g') or as $(-g', -g)$) in the circular ordering of each of the species under consideration, then “fuse” g and g' into a single replacement *metagene*. (Thus the size of \mathcal{G} , and the length of the circular orderings, is effectively decreased by one.)

The authors of [3.40] report that this single reduction rule yields a speedup by a factor of 6 over the implementation of Blanchette, Bourque and Sankoff [3.10].

This first phase of kernelization on a Campanulaceae data set of 13 species with an initial set of genes \mathcal{G} of size $|\mathcal{G}| = 150$ was found to simplify the input to a set \mathcal{G}' of metagenes of size $|\mathcal{G}'| = 35$. Following this initial kernelization, the second phase considers a sampling of all $(2n - 5)!!$ leaf-labelled trees on n leaves, and for each of these uses a separate heuristic to explore possible internal labellings.

This heuristic was not developed as an *FPT* algorithm, yet it is one, for a realistic natural parameter — the total cost of the tree.

Theorem 3.3.1. *The BREAKPOINT MEDIAN PROBLEM is fixed-parameter tractable for the parameter k taken to be the total cost of the tree.*

Proof. If the tree has total cost k then it has at most k internal edges (since each contributes some cost to the total) and therefore at most $k - 1$ leaves and at most $k - 2$ internal vertices. It is also not hard to see that after kernelization, the number of genes in each sequence is at most k . Thus, the cost of exhaustively checking each of the $(2k - 7)!!$ leaf-labelled trees on $k - 1$ leaves, and exhaustively trying all possible assignments of the $k!$ possible gene orderings (of the kernelized instance) to the internal vertices of the trees — all of this can be accomplished in time bounded by a function of k . \square

The running time of the kernelization + brute force *FPT* algorithm described above would not seem conducive to practical implementation, since the implicit parameter function $f(k)$ is around $(k!)^k$, which exceeds 10^{20} when $k = 4$. What Moret *et al.* have implemented is essentially a heuristic adaptation of this *FPT* algorithm, based on a sampling of the possible trees and a sampling of the possible internal vertex assignments.

General heuristic design strategies that correspond to some of the main *FPT* methods are displayed in Table 3.3. The essential theme is to obtain heuristic methods from *FPT* algorithms by strategies for *deflating the parametric costs* by truncating or sampling the search trees or obstruction sets, etc.

Table 3.2. Some *FPT* methods and heuristic strategies

FPT Technique	Heuristic Design Strategy
Reduction to a Problem Kernel	A useful pre-processing subroutine for any heuristic.
Search Tree	Explore only an affordable, heuristically chosen subtree.
Well-Quasiordering	Use a sample of the obstruction set.
Color-Coding	Use a sample of the hash functions.

The following problem is also fixed-parameter tractable.

THE STEINER PROBLEM FOR GENERALIZED HYPERCUBES

Instance: The input the problem consists of the following pieces of information:

1. A set of complete weighted digraphs D_i for $i = 1, \dots, n$, each described by a set of vertices V_i and a function

$$t_i : V_i \times V_i \rightarrow \mathbb{N}$$

(We refer to the vertices of D_i as *character states*, to D_i as the *character state digraph*, and to t_i as the *state transition cost function* for the i th character.)

2. A positive integer k_1 such that $|V_i| \leq k_1$ for $i = 1, \dots, n$.
3. A set X of k_2 length n vectors x_j for $j = 1, \dots, k_2$, where the i th component $x_j[i] \in V_i$. That is, for $j = 1, \dots, k_2$,

$$x_j \in \Omega = \prod_{i=1}^n V_i$$

4. A positive integer M .

Parameter: (k_1, k_2)

Question: Is there a rooted tree $T = (V, E)$ and an assignment to each vertex $v \in V$ of T of an element $y_v \in \Omega$, such that:

- X is assigned one-to-one to the set of leaves of T ,
- The sum over all parent-child edges uv of T , of the *total transition cost* for the edge, defined to be

$$\sum_{i=1}^n t_i(y_u[i], y_v[i])$$

is bounded by M ?

Theorem 3.3.2. THE STEINER PROBLEM FOR GENERALIZED HYPERCUBES is *fixed-parameter tractable*.

Proof. We define an equivalence relation $i \sim j$ on the index space $\{1, \dots, n\}$ that allows us to combine D_i and D_j and obtain an equivalent smaller instance. In order to define \sim we first define some other equivalences.

Fix $m \leq k_1$ and let l be an integer edge labeling of the complete digraph K_m on m vertices. Let v_1, \dots, v_m denote the vertices of K_m . Let T be a rooted tree with k_2 leaves labeled from v_1, \dots, v_m . Define the *cost* of T with respect to l to be the minimum, over all possible labelings s of the internal vertices of T with labels taken from $\{v_1, \dots, v_m\}$, of the sum over the parent-child edges of T of the transition costs given by l on the labels, and write this as

$$\text{cost}(T, l) = \min_s \{\text{cost}(T, s, l)\}$$

If l and l' are integer edge labelings of K_m and T is as above, then define $l \sim_T l'$ if and only if $\exists s$ such that

$$\text{cost}(T, l) = \text{cost}(T, s, l) = \text{cost}(T, s, l') = \text{cost}(T, l')$$

and define $l \sim l'$ if and only if $l \sim_T l'$ for all such trees T .

For $i, i' \in \{1, \dots, n\}$ define $i \sim i'$ if and only if:

1. $|V_i| = |V_{i'}| = m$ so that the only difference between D_i and $D_{i'}$ is in their arc-labelings l and l' , and
2. $l \sim l'$.

The kernelization algorithm can now be described quite simply. Let \mathcal{I} be an instance of the problem. If there are indices $i \neq i'$ for which $i \sim i'$, then modify \mathcal{I} by combining these into one character state digraph with the state transition cost function given by the arc-labeling given $l + l'$, where these are the cost functions for D_i and $D_{i'}$, respectively. Let \mathcal{I}' denote the modified instance.

The correctness of the reduction to the smaller instance is obvious. We need only to note that the equivalence $i \sim i'$ can be determined in time bounded by a function of the parameter and that number of equivalence classes is similarly bounded by a function of the parameter. \square

The parameter function for this simple kernelization-based *FPT* algorithm is nearly as discouraging as the one for *BREAKPOINT PHYLOGENY*. We remark that most of the expense is in determining when two transition digraph indices i and i' are equivalent by testing them on all possible trees with k_2 leaves. This suggests a heuristic algorithm that combines indices when they fail to be distinguished by a (much smaller) random sample of trees and leaf-labelings.

A previous survey described this *FPT* result in the context of an encounter with an evolutionary biologist who reported earlier, rather fruitless interactions with theoretical computer scientists who proved that his problems were *NP*-complete and “went away”. We claimed that we were different! and that

we had a result on one of his computational problems (THE STEINER PROBLEM FOR HYPERCUBES) that might be of interest. After we described the *FPT* algorithm he said simply [3.27]:

“That’s what I already do!”

Those who design *FPT* algorithms should keep in mind that their $f(k)$ ’s are only the best they are able to prove concerning a worst-case analysis, and that their algorithms may in fact be much more useful in practice than the pessimistic analysis indicates, on realistic inputs, particularly if any nontrivial kernelization is involved. Furthermore, large parametric costs can also be systematically mitigated in heuristic adaptations of *FPT* algorithms. Real usefulness can only be settled by implementation and experimentation.

3.4 A Critical Tool for Evaluating Approximation Algorithms

The emphasis in the substantial new industry of research on polynomial-time approximation algorithms is concentrated on the notions of:

- Polynomial-time constant factor approximation algorithms.
- Polynomial-time approximation schemes.

The connections between the *parameterized complexity* and *polynomial-time approximation* programs are deep and developing rapidly. 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 [3.4] 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 [3.16] 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 [3.53] has a running time of $O(n^{2\lceil 2/\epsilon \rceil - 2})$. For a 20% error, we thus have a “polynomial” running time of $O(n^{18})$.
- The PTAS for the UNBOUNDED BATCH SCHEDULING problem due to Deng, Feng, Zhang and Zhu [3.20] 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 [3.36] has a running time of $O(n^{8(1+(2/\epsilon))})$; thus we have $O(n^{88})$ time for a 20% error.

logic of an optimization problem is what allows PTASs to be developed [3.37]. 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.

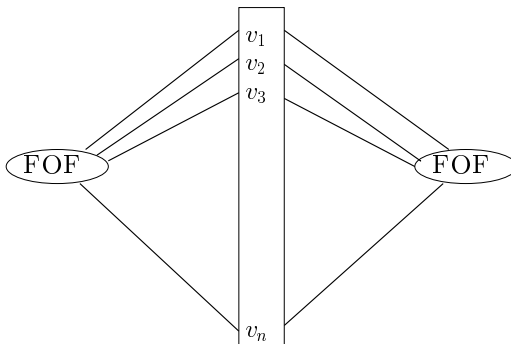
The following theorem is recent joint work with Cai, Juedes and Rosamond [3.12].

Theorem 3.4.2. *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 $b_i = [v_{i,1}, \dots, v_{i,n}]$ will be set to true. We interpret this event as “vertex j is the i th 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 $\bigvee_{j=1}^n v_{i,j}$. 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 $\bigvee_{(u,v) \in E} v_{i,u} v_{j,v}$. Each FOF $f_{i,j}$ ensures that there is an edge in G between the i th vertex the clique and the j th vertex in the clique.

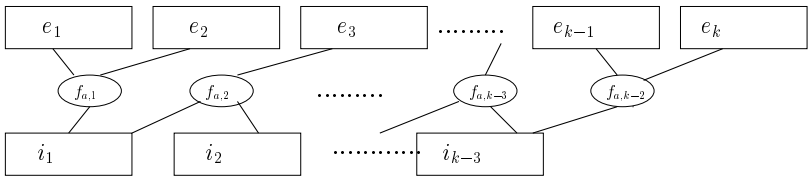
It is straightforward to show that $C' = \{f_1, \dots, f_k, f_{1,2}, \dots, f_{k-1,k}\}$ has a weight k satisfying assignment if and only if G has a clique of size k . To see this, notice that any weight k satisfying assignment for C' must satisfy exactly one variable in each block b_i . Each first order formula $f_{i,j}$ ensures that there is an edge between the i th vertex in the potential clique and the j th 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 i th vertex in the potential clique is not the j th 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 four. 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 $e_i = [e_{i,1}, \dots, e_{i,n}]$ of variables is connected to exactly one FOF inside the widget. Each internal block $i_j = [i_{j,1}, \dots, e_{j,n}]$ is connected to exactly two FOFs inside the widget. The $k - 2$ FOFs are given as follows. The FOF $f_{a,1}$ is $\bigvee_{j=1}^n e_{1,j} e_{2,j} i_{1,j}$. For each $2 \leq l \leq k - 3$, the FOF

$$f_{a,l} = \bigvee_{j=1}^n i_{l-1,j} e_{l+1,j} i_{l,j}. \text{ Finally, } f_{a,k-2} = \bigvee_{j=1}^n i_{k-3,j} e_{k-1,j} e_{k,j}. \text{ These } k - 2$$

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.

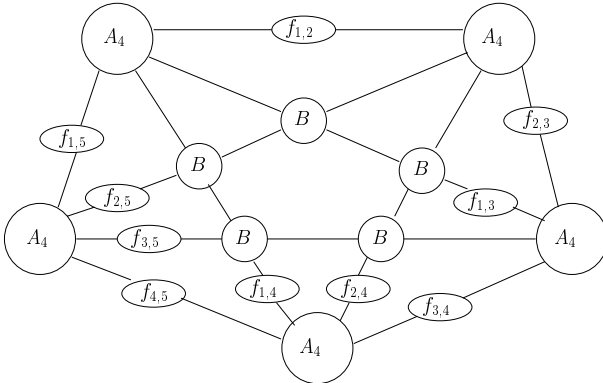


Since each internal block is connected to exactly two FOFs, the incidence graph for this widget can be drawn on the plane without crossing any edges.

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 $u = [u_1, \dots, u_n]$ and $v = [v_1, \dots, v_n]$, insert a widget B that introduces two new blocks of n variables $u_1 = [u_{1_1} \dots u_{1_n}]$ and $v_1 = [v_{1_1} \dots v_{1_n}]$ 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 two 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 be shown to be $W[1]$ -hard. PTAS's for these problems therefore can never be useful, since the goodness of the approximation must be paid for in the exponent of the polynomial running time. A PTAS result alone establishes that an approximation problem is in the parameterized complexity class XP . By analogy, one would not reasonably claim any practical significance for a demonstration that a problem just belongs to NP . It would be interesting to sort out which problems with PTAS's have any hope of practical approximation, and for which such "good news" (see [3.7] for a comprehensive survey) is chimerical.

3.5 The Extremal Connection: A General Method Relating *FPT*, Polynomial-Time Approximation, and Pre-Processing Based Heuristics

The toolkit for establishing fixed-parameter tractability includes a number of mathematically deep methods: well-quasi-ordering, color-coding, and bounded treewidth — as well as the elementary methods of kernelization and search trees. Some of these positive methods are very powerful at *classifying* problems as fixed-parameter tractable, but are far from any practical significance (for example, methods based on well-quasiordering). For the purposes of practical algorithm design, *reduction to a problem kernel* is probably the single most important contribution to the systematic design of heuristics. This is in some sense a comprehensive connection, since *fixed-parameter tractability* is equivalent to *kernelizability* as shown by Lemma 3.3.1

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 four, for another example, includes the following nontrivial transformation [3.24]. Suppose G has a vertex x of degree three 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 pre-processing 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 [3.3], the effort to kernelize is amply rewarded by the subsequently exponentially smaller search tree. Similar results have also been reported by Moret *et al.* with respect to the BREAKPOINT PHYLOGENY problem [3.40].
- (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 [3.15]. 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 3.3.1 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 [3.17, 3.26, 3.2].)

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?

The technique of *coordinatized kernelization* is aimed at these difficulties, and we will illustrate it by example 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 structure 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 two 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 two, 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 one, two, or three 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 one, 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 two 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 three 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 $5.75k$. \square

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.

References

- 3.1 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* 73:235–276, 1995.
- 3.2 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'02)*. Springer Lecture Notes in Computer Science, 2002.
- 3.3 J. Alber, J. Gramm, and R. Niedermeier. Faster exact algorithms for hard problems: a parameterized point of view. *Discrete Mathematics* 229:3–27, 2001.
- 3.4 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 (FOCS'96)*, pages 2–11, 1996.
- 3.5 S. Arora. Nearly linear time approximation schemes for Euclidean TSP and other geometric problems. In *Proceedings of the 38th Annual IEEE Symposium on the Foundations of Computer Science (FOCS'97)*, pages 554–563, 1997.
- 3.6 V. Arvind, M. R. Fellows, M. Mahajan, V. Raman, S. S. Rao, F. A. Rosamond, and C. R. Subramanian. Parametric duality and fixed parameter tractability. Manuscript, 2001.

- 3.7 G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation*. Springer-Verlag, Heidelberg, 1999.
- 3.8 N. Bansal and V. Raman. Upper bounds for MAXSAT: further improved. In *Proceedings of the 10th International Symposium on Algorithms and Computation (ISAAC'99)*. Springer Lecture Notes in Computer Science 1741, pages 247–258, 1999.
- 3.9 C. Bazgan. Schémas d'approximation et complexité paramétrée. Rapport de stage de DEA d'Informatique à Orsay, 1995.
- 3.10 M. Blanchette, G. Bourque, and D. Sankoff. Breakpoint phylogenies. In S. Miyano and T. Tagaki, editors, *Genome Informatics 1997*, Universal Academy Press, Tokyo, 1997, pages 25–34.
- 3.11 L. Cai. The complexity of coloring parameterized graphs. To appear in *Discrete Applied Mathematics*.
- 3.12 L. Cai, M. Fellows, D. Juedes, and F. Rosamond. Efficient polynomial-time approximation schemes for problems on planar structures: upper and lower bounds. Manuscript, 2001.
- 3.13 M. Cesati and L. Trevisan. On the efficiency of polynomial time approximation schemes. *Information Processing Letters* 64:165–171, 1997.
- 3.14 M. Cesati and H. T. Wareham. Parameterized complexity analysis in robot motion planning. In *Proceedings of the 25th IEEE International Conference on Systems, Man and Cybernetics: Volume 1*. IEEE Press, Los Alamitos, CA, pages 880–885, 1995.
- 3.15 P. Cheeseman, B. Kanefsky, and W. Taylor. Where the really hard problems are. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 331–337, 1991.
- 3.16 C. Chekuri and S. Khanna. A PTAS for the multiple knapsack problem. Manuscript, 2000.
- 3.17 J. Chen, I. A. Kanj, and W. Jia. Vertex cover: further observations and further improvements. In *Proceedings of the 25th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'99)*. Springer Lecture Notes in Computer Science 1665, pages 313–324, 1999.
- 3.18 J. Chen and A. Miranda. A polynomial-time approximation scheme for general multiprocessor scheduling. In *Proceedings of the 31st Annual ACM Symposium on the Theory of Computing (STOC'99)*, pages 418–427, 1999.
- 3.19 F. Dehne, A. Rau-Chaplin, U. Stege, and P. Taillon. Solving large FPT problems on coarse grained parallel machines. Manuscript, 2001.
- 3.20 X. Deng, H. Feng, P. Zhang, and H. Zhu. A polynomial time approximation scheme for minimizing total completion time of unbounded batch scheduling. In *Proceedings of the 12th International Symposium on Algorithms and Computation (ISAAC'01)*. Springer Lecture Notes in Computer Science 2223, pages 26–35, 2001.
- 3.21 R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer-Verlag, Heidelberg, 1998.
- 3.22 R. G. Downey and M. Fellows. Parameterized complexity after almost ten years: review and open questions. In *Proceedings of Combinatorics, Computation and Logic, DMTCS'99 and CATS'99*, Australian Computer Science Communications, Springer-Verlag, Singapore, vol. 21, pages 1–33, 1999.
- 3.23 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, Heidelberg, pages 194–213, 1997.

- 3.24 R. G. Downey, M. R. Fellows, and U. Stege. Parameterized complexity: a framework for systematically confronting computational intractability. In R. Graham, J. Kratochvíl, J. Nešetřil, and F. Roberts, editors, *Proceedings of the DIMACS-DIMATIA Workshop*, Prague, 1997. AMS-DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 49, pages 49–99, 1999.
- 3.25 T. Erlebach, K. Jansen, and E. Seidel. Polynomial time approximation schemes for geometric graphs. In *Proceedings of the 12th Annual Symposium on Discrete Algorithms (SODA'01)*, pages 671–679, 2001.
- 3.26 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. In *Proceedings of the 20th FSTTCS Conference*. Springer Lecture Notes in Computer Science 1974, pages 240–251, 2000.
- 3.27 J. Felsenstein. Private communication, 1997.
- 3.28 M. Galota, C. Glasser, S. Reith, and H. Vollmer. A polynomial time approximation scheme for base station positioning in UMTS networks. In *Proceedings of Discrete Algorithms and Methods for Mobile Computing and Communication*, 2001.
- 3.29 M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco, 1979.
- 3.30 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 *Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'99)*, Springer Lecture Notes in Artificial Intelligence 1730, pages 1–18, 1999.
- 3.31 J. Gramm and R. Niedermeier. Faster exact algorithms for MAX2SAT. In *Proceedings of the 4th Italian Conference on Algorithms and Complexity*. Springer Lecture Notes in Computer Science 1767, pages 174–186, 2000.
- 3.32 M. Grohe. Generalized model-checking problems for first-order logic. In *Proceedings of the 18th Annual Symposium on Theoretical Aspects of Computer Science (STACS'01)*. Springer Lecture Notes in Computer Science 2010, pages 12–26, 2001.
- 3.33 M. Grohe. The parameterized complexity of database queries. In *Proceedings of the 20th ACM symposium on Principles of Database Systems (PODS'01)*, ACM Press, pages 82–92, 2001.
- 3.34 M. Hallett, G. Gonnet, and U. Stege. Vertex cover revisited: a hybrid algorithm of theory and heuristic. Manuscript, 1998.
- 3.35 F. Henglein and H. G. Mairson. The complexity of type inference for higher-order typed lambda calculi. In *Proceedings of the 18th Annual ACM Symposium on Principles of Programming Languages (POPL'91)*, pages 119–130, 1991.
- 3.36 Y. Karuno and H. Nagamochi. A polynomial time approximation scheme for the multi-vehicle scheduling problem on a path with release and handling times. In *Proceedings of the 12th International Symposium on Algorithms and Computation (ISAAC'01)*. Springer Lecture Notes in Computer Science 2223, 36–47, 2001.
- 3.37 S. Khanna and R. Motwani. Towards a syntactic characterization of PTAS. In *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing (STOC'96)*, pages 329–337, 1996.

- 3.38 S. Khot and V. Raman. Parameterized complexity of finding subgraphs with hereditary properties. In *Proceedings of the 6th Annual International Computing and Combinatorics Conference (COCOON'00)*. Springer Lecture Notes in Computer Science 1858, pages 137–147, 2000.
- 3.39 O. Lichtenstein and A. Pnueli. Checking that finite-state concurrents programs satisfy their linear specification. In *Proceedings of the 12th ACM Symposium on Principles of Programming Languages (POPL'85)*, pages 97–107, 1985.
- 3.40 B. Moret, S. Wyman, D. Bader, T. Warnow, and M. Yan. A new implementation and detailed study of breakpoint analysis. In *Proceedings of the 6th Pacific Symposium on Biocomputing (PSB'01)*, pages 583–594, 2001.
- 3.41 P. Moscato. Controllability, parameterized complexity, and the systematic design of evolutionary algorithms. Manuscript, 2001. See <http://www.densis.fee.unicamp.br/~moscato>
- 3.42 R. Niedermeier. Some prospects for efficient fixed-parameter algorithms. In *Proceedings of the 25th Conference on Current Trends in Theory and Practice of Informatics (SOFSEM'98)*. Springer Lecture Notes in Computer Science 1521, pages 168–185, 1998.
- 3.43 R. Niedermeier and P. Rossmanith. An efficient fixed parameter algorithm for 3-hitting set. *Journal of Discrete Algorithms* 2(1), 2001.
- 3.44 C. Papadimitriou and M. Yannakakis. On the complexity of database queries. In *Proceedings of the 16th ACM Symposium on Principles of Database Systems (PODS'97)*, pages 12–19, 1997.
- 3.45 I. Pe'er and R. Shamir. The median problems for breakpoints are NP-complete. *Electronic Colloquium on Computational Complexity Technical Report* 98-071, <http://www.ecc.uni-trier.de/eccc>.
- 3.46 V. Raman. Parameterized complexity. In *Proceedings of the 7th National Seminar on Theoretical Computer Science*, Chennai, India, pages 1–18, 1997.
- 3.47 H. Shachnai and T. Tamir. Polynomial time approximation schemes for class-constrained packing problems. In *Proceedings of the 3rd International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX'00)*. Springer Lecture Notes in Computer Science 1913, pages 144–154, 2000.
- 3.48 R. Shamir and D. Tzur. The maximum subforest problem: approximation and exact algorithms. In *Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms (SODA'98)*, pages 394–399, 1998.
- 3.49 U. Stege. Resolving conflicts in problems in computational biochemistry. Ph.D. dissertation, ETH, 2000.
- 3.50 M. Truszczynski. On Computing Large and Small Stable Models. In *Proceedings of the International Conference on Logic Programming*, pages 169–183, 1999. Full version to appear in *Journal of Logic Programming*.
- 3.51 K. Weihe. Covering trains by stations, or the power of data reduction. In *Proceedings of the 1st Workshop on Algorithm Engineering and Experiments (ALENEX'98)*. Springer Lecture Notes in Computer Science 1619, pages 1–8, 1998.
- 3.52 K. Weihe. On the differences between practical and applied. Dagstuhl Workshop on Experimental Algorithmics, September 2000.
- 3.53 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. In *Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms (SODA'98)*, pages 21–32, 1998.