Parameterized Complexity: A Framework for Systematically Confronting Computational Intractability

Rodney G. Downey, Michael R. Fellows, and Ulrike Stege

Abstract. In this paper we give a programmatic overview of parameterized computational complexity in the broad context of the problem of coping with computational intractability. We give some examples of how fixed-parameter tractability techniques can deliver practical algorithms in two different ways: (1) by providing useful exact algorithms for small parameter ranges, and (2) by providing guidance in the design of heuristic algorithms. In particular, we describe an improved *FPT* kernelization algorithm for Vertex Cover, a practical *FPT* algorithm for the Maximum Agreement Subtree (MAST) problem parameterized by the number of species to be deleted, and new general heuristics for these problems based on *FPT* techniques. In the course of making this overview, we also investigate some structural and hardness issues. We prove that an important naturally parameterized problem in artificial intelligence, STRIPS Planning (where the parameter is the size of the plan) is complete for $W[1]$. As a corollary, this implies that k-Step Reachability for Petri Nets is complete for $W[1]$. We describe how the concept of *treewidth* can be applied to STRIPS Planning and other problems of logic to obtain *FPT* results. We describe a surprising structural result concerning the top end of the parameterized complexity hierarchy: the naturally parameterized Graph k-Coloring problem cannot be resolved with respect to *XP* either by showing membership in *XP*, or by showing hardness for *XP* without settling the $P = NP$ question one way or the other.

1. Introduction

There are basically two different ways that one can view the theory of parameterized complexity. The first way, and the one that is easiest to arrive at, is as a kind of first aid that can sometimes be applied to problems that are *NP*-hard, *PSPACE*-hard or undecidable. That is, it can be viewed as a potential means of coping with *classical* intractability.

The second way that one can view parameterized complexity is as a fundamentally richer and generally more productive *primary framework* for problem analysis and algorithm design, including the design of heuristic and approximation algorithms. For the moment, we will concentrate on the first point of view, and spend some time considering the current situation of our collective efforts to "cope with intractability" in the words of Garey and Johnson [**GJ79**], Chapter 6. In the concluding section we will summarize the arguments for the second point of view.

It is difficult to discuss the current intellectual situation with respect to theoretical computer science without taking note of the unhappiness, in some sense, of the audience of our efforts to understand complexity and deliver useful products. For some time, the practical folks portrayed in the amusing cartoons of Garey and Johnson [**GJ79**], Chapter 1, have been somewhat disappointed with theoretical computer science, and there have been numerous incidents of "restless drums" in the jungles of computer applications. Such grumblings may have something to do with the recent spate of soul-searching among theorists [**HL92, Hart94, PGWRS96, AFGPR96**]. We mention two recent examples.

EXAMPLE 1.1 (Yet Another Call for Reform). One of the plenary addresses at the AAAI meeting in 1996 was concerned with the broad theme of how computer science practice and theory interact [**DKL96**]. The discussion in [**DKL96**] can be summarized as:

1. Pointing to a particular problem, STRIPS PLANNING, as central to the entire field of artificial intelligence.
2. Proposing that practitioners and theorists collaborate in an intense analysis of this one problem, to understand what makes it hard, and to come up with something more useful than a *PSPACE*-completeness classification.
3. Suggesting that the usual framework for concrete complexity analysis is wrong-headed, historically contingent, unnatural (especially worst-case asymptotic analysis), and reflects an unhappy state of interaction between computer science theory and practice.

Regarding the third point, most theorists have probably heard similar charges and complaints from colleagues in applied areas of computer science in their own departments.

EXAMPLE 1.2 (An Encounter With a Computational Biologist). In recent conversations with a biologist who is heavily involved in combinatorial computing [**Fel97**], the following summary was offered of his interaction with theoretical computer scientists.

> *"About ten years ago some computer scientists came by and said they had heard that we have some really cool problems. They showed that the problems are* NP*-complete and went away!"*

We might comment that if this interaction had been more recent, then perhaps the computer scientists would also have proved that the problems are unlikely to have efficient approximation algorithms.

The Pervasive Nature of Computational Intractability and Various Coping Strategies. Arguably the most fundamental discovery of the first decades of theoretical computer science is that most computational problems are *hard* in a variety of mathematically interesting ways. Computer science practitioners quite naturally would like to shoot the messenger who brings so much bad news! In this difficult situation, computer science theory has articulated a few general programs for systematically coping with the ubiquitous phenomena of computational intractability. We list these basic approaches:

- The idea of focusing on average-case as opposed to worst-case analysis of problems.
- The idea of settling for approximate solutions to problems, and of looking for efficient approximation algorithms.
- The idea of using randomization in algorithms.
- The idea of harnessing quantum mechanics, or biomolecular chemistry, to create qualitatively more powerful computational mechanisms.

To this list of basic mathematical strategies for coping with intractability, we argue should be added:

- The idea of devising *FPT* algorithms for parameterizations of a problem.

A list such as this cannot be considered complete without including another coping strategy that mathematical theorists have frequently contributed to. This approach antedates modern complexity-theoretic frameworks and persists with great strength as a kind of "old religion" among many practitioners. It has also recently regained some respectability among theorists as reflected in the DIMACS Challenges [**JM93, JT96**] and in new journals such as the *Journal of Heuristics* and the ACM *Journal of Experimental Algorithms*):

- The design of mathematically informed, but perhaps unanalyzable heuristics, that are empirically evaluated by their performance on sets of benchmark instances.

The actual state of the practical world of computing is that (with the exception of some areas) there is not much systematic connection to work in theoretical computer science on algorithms and complexity. Overwhelmingly, in fact, it is heuristic algorithms that are relied on to deal with the problems encountered in most applications.

2. Fixed-Parameter Tractability

The basic concept of the parameterized complexity framework is that of *fixed-parameter tractability*. It is the notion of good complexity behaviour from which all other aspects of the theory follow. The definition is best introduced through concrete examples.

VERTEX COVER
Instance: A graph $G = (V, E)$ and a positive integer k.
Parameter: k
Question: Does G have a vertex cover of size k? That is, is there a subset $V' \subseteq V$ of size at most k such that for every edge $uv \in E$, either $u \in V'$ or $v \in V'$?

DOMINATING SET
Instance: A graph $G = (V, E)$ and a positive integer k.
Parameter: k
Question: Does G have a dominating set of size k? That is, is there a subset $V' \subseteq V$ of size at most k such that every vertex $u \in V$ of G either belongs to V' or has a neighbor in V'?

THE STEINER PROBLEM FOR HYPERCUBES
Instance: A set $S = \{x_i : 1 \leq i \leq k\}$ of binary vectors, $x_i \in \{0,1\}^n$ for $i = 1, ..., k$, and a positive integer m.
Parameter: k
Question: Is there a tree $T = (V, E)$ and a labeling of the vertices of T with elements of $\{0,1\}^n$ such that the following conditions are satisfied? (1) The leaves are labeled 1:1 with the elements of S. (2) The sum over the edges uv of T of the Hamming distance between the labels $l(u) \in \{0,1\}^n$ and $l(v) \in \{0,1\}^n$ is at most m.

THE MAXIMUM AGREEMENT SUBTREE PROBLEM (MAST)
Instance: A set of rooted trees $T_1, ..., T_r$ ($r \geq 3$) with the leaf set of each T_i labeled 1:1 with a set of species X, and a positive integer k.
Parameter: k
Question: Is there a subset $S \subseteq X$ of size at most k such that T_i restricted to the leaf set $X' = X - S$ is the same (up to label-preserving isomorphism and ignoring vertices of degree 2) for $i = 1, ..., r$?

All of these problems are *NP*-complete ([**GJ79, AK94**]) and are described above in the standard way for the parameterized complexity framework. Part of the input (which may be some aggregate of various aspects of the input) is identified as the *parameter* for the problem specification. (In order to consider a parameterized problem classically, just ignore the parameter part of the specification.) All of these problems can be solved in time $O(n^{f(k)})$ by simple brute force algorithms. For example, for VERTEX COVER and DOMINATING SET we can simply try all k-subsets of vertices.

For VERTEX COVER we can do qualitatively better. Papadimitriou and Yannakakis showed that VERTEX COVER can be solved in time $O(3^k n)$ [**PY96**]. Balasubramanian, Fellows and Raman gave an algorithm with running time $O((53/40)^k k^2 + kn)$ [**BFR98**]. In §4 we describe a new and relatively simple *FPT* algorithm that improves on this. Note that since the exponential dependence on the parameter k in the last expression is *additive*, VERTEX COVER is *well-solved for input of any size so long as k is no more than around 60.* The difference between the complexities of DOMINATING SET and VERTEX COVER is displayed in Table 1.

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

In parameterized complexity, the process of interviewing practice in order to formulate the fundamental object of study is enriched. Besides specifying the input to the problem, we specify distributional aspects that may belong to some limited range of values for which an exponential contribution to overall problem complexity may be acceptable. This distributional information, which may be an aggregate of factors, is codified as the *parameter*. A single classical problem may thus shatter into many different associated *parameterized problems*.

EXAMPLE 2.1 (THE STEINER PROBLEM FOR HYPERCUBES). This problem is of interest to biologists in the computation of phylogenetic trees under the criterion of minimum evolution / maximum parsimony [**SOWH96**]. The set S corresponds to a set of species, and the binary vectors correspond to information about the species, each component recording the answer to some question (as 0 or 1), such as: "Does it have wings?" or "Is there a thymine at a certain position in the DNA sequence?" Each such bit of information is termed a *character* of the species. In realistic applications the number k of species may usefully be around 40 or 60, while the number of characters n may be very large.

How Parameters Naturally Arise. Most computational problems involve several pieces of input, one or more of which may be a relevant parameter for various applications.

- *Graph linear layout width metrics* are of interest in VLSI layout and routing problems and have important applications for width values of $k \leq 10$. Interval graphs of pathwidth $k \leq 10$ have applications in DNA sequence reconstruction problems [**BDFHW95**].
- *Logic and database problems* frequently are defined as having input consisting a formula (which may be small and relatively invariant), and some other structure (such as a database) which is typically quite large and changeable. Formula size, or other aspects of formula structure may be a relevant parameter [**Yan95**].

- *Hardware constraints* are a common source of natural parameters. The number of processors or machines to be scheduled may be bounded by a value such as $k \leq 20$. In [**AMOV91**] it was proposed to streamline chip implementations of cryptosystems by bounding the Hamming weight of keys. The number of degrees of freedom in a robot motion-planning problem is commonly in the range $k \leq 10$ [**CW95**]. The number of wiring layers in VLSI chip manufacture is typically bounded by $k \leq 30$ [**Len90**].
- *Network problems* may be naturally concerned with optimally locating a small number of facilities.

These few examples are only suggestive and by no means exhaustive. There are myriad ways in which numbers that are small or moderately large (e.g., $k \leq 40$) arise naturally in problem specifications. Important distributional parameters may also arise in ways that are not at all obvious. For an example of this sort, Thorup has recently shown that the flow graphs of structured programs for the major computer languages have treewidth $k \leq 7$ [**Th97**]. Graphs of pathwidth bounded by a similar number have been used in modeling dependencies in sentences of natural languages [**KT92**]. Attention to hidden parameters can sometimes explain why a problem is easier to solve in practice than *NP* or *PSPACE* hardness results would suggest.

The Basic Definitions. The basic definitions of parameterized complexity are as follows.

Definition 2.1. A *parameterized language* L is a subset $L \subseteq \Sigma^* \times \Sigma^*$. If L is a parameterized language and $(x, y) \in L$ then we will refer to x as the *main part*, and refer to y as the *parameter*. It makes no difference to the theory and is occasionally more convenient to consider that y is an integer, or equivalently to define a parameterized language to be a subset of $\Sigma^* \times I\!N$.

Definition 2.2. A parameterized language L is *fixed-parameter tractable* if it can be determined in time $f(k)n^\alpha$ whether $(x, k) \in L$, where $|x| = n$, α is a constant independent of both n and k and f is an arbitrary function. The family of fixed-parameter tractable parameterized languages is denoted *FPT*.

It is somewhat surprising, although the argument is not hard, that *FPT* is unchanged if the definition above is modified by replacing $f(k)n^\alpha$ by $f(k) + n^\alpha$ [**CCDF97**].

About half of the naturally parameterized problems cataloged as *NP*-complete in the book by Garey and Johnson [**GJ79**] are in *FPT*, including three of the six basic problems singled out for attention in Chapter 3.

It is always possible to parameterize a problem in various ways that are fixed-parameter tractable, yet it is not surprising that many parameterized problems apparently do not belong to *FPT*. The naturally parameterized Dominating Set problem defined above is one of these. Just as with the issue of polynomial-time complexity, we can find evidence for fixed-parameter *intractability* by studying the appropriate notion of problem transformation.

DEFINITION 2.3. 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.

EXAMPLE 2.2 (An Illustrative Non-Example). In first examining the notion of a parametric transformation it can be helpful to see how they differ from ordinary polynomial-time reductions. Recall that for a graph $G = (V, E)$ on n vertices, a set of vertices $V' \subseteq V$ is a k-clique in G if and only if $V - V'$ is a vertex cover in the complementary graph G' where vertices are adjacent if and only if they are not adjacent in G. This gives an easy polynomial-time reduction of the naturally parameterized CLIQUE problem to the naturally parameterized VERTEX COVER problem, transforming the instance (G, k) of CLIQUE into the instance (G', k') of VERTEX COVER. But this is not a parametric transformation, since $k' = n - k$ is not purely a function of k. The evidence is that there is no parametric transformation in this direction between these two problems (although there is a parametric transformation in the reverse direction, either trivially, since VERTEX COVER is in *FPT*, or nontrivially by the construction described in [**DF95b**]).

EXAMPLE 2.3 (An Illustrative Example). There is a fairly elaborate parametric transformation from the naturally parameterized CLIQUE problem to the naturally parameterized DOMINATING SET problem, mapping (G, k) to (G', k') where $k' = 2k$ [**DF95a, DF98**]. The evidence is that there is no such parametric transformation in the other direction.

The essential property of parametric transformations is that if L transforms to L' and $L' \in FPT$, then $L \in FPT$. This naturally leads to a completeness program based on a hierarchy of parameterized problem classes:

$$FPT \subseteq W[1] \subseteq W[2] \subseteq \cdots \subseteq W[SAT] \subseteq W[P] \subseteq AW[P] \subseteq XP$$

The parameterized analog of *NP* is $W[1]$, and $W[1]$-hardness is the basic evidence that a parameterized problem is likely not to be fixed-parameter tractable. The k-STEP HALTING PROBLEM FOR NONDETERMINISTIC TURING MACHINES is $W[1]$-complete [**CCDF97**]. Since the $q(n)$-STEP HALTING PROBLEM is essentially the defining problem for *NP*, the analogy is very strong. The reader will find more about parametric intractability in §6.

How Compelling is Fixed-Parameter Tractability? The notion of *fixed-parameter tractability* is the basic concept of the theory — but how good, really, is this notion of *good complexity behaviour*? It might be objected that Table 1 is misleading, unless *FPT* parameter functions such as 2^k are typical. Certainly functions such as $2^{2^{2^k}}$ are allowed by the definition, and would be impractical for $k = 3$, which suggests that the basic definition allows too much pathology.

There are two main responses. First of all, we are already used to some risk-taking in definitions, since the notion of polynomial time allows for, e.g., $O(n^{10})$, which is impractical. A parameterized problem is just an ordinary problem for which some aspect of the input has been designated as the parameter. Ignoring the parameter, if the problem can be solved in polynomial time, that is, in time polynomial in both the total input size n and the parameter k, then trivially this is an *FPT* algorithm. In other words, considered classically, *FPT* is a superset of *P*, and it is intended to be a generalization that allows us to do something for problems that are not in *P* and that may even be *PSPACE* hard or undecidable. We have to expect to risk something in formulating such a general attack on intractability. The definition simply asks whether the difficulty of the problem can be confined to a function of the parameter, with the other costs being polynomial. How else would one naturally formulate a generalization of *P* having these kinds of ambitions?

The second response is that there are many examples, other than Vertex Cover, suggesting that "reasonable" (e.g., single exponential) parameter functions are frequently obtainable for natural problems (possibly after some rounds of improvement). For example, consider the problem Maximum Satisfiability where the parameter k denotes the number of clauses to be satisfied. This was shown by Cai and Chen [**CC97**] to be in *FPT* with the parameter function 2^{2ck}, when the clause size is bounded by c. The parameter function was improved by Mahajan and Raman [**MR98**] to ϕ^k (without assuming a bound on the clause size), where ϕ is the golden ratio $(1+\sqrt{5})/2$. Franco and others in [**Fr97**] have shown that the falsifiability problem for pure implicational formulas having k negations is in *FPT* with parameter function k^k. (Can this be improved to 2^k?) Although the type checking problem for the programming language ML is *PSPACE*-complete [**HM91**], this is handled in implementations in linear time with a parameter function of 2^k, where k is the nesting depth of *let*'s, a very natural parameter for this problem (one that explains why the problem did not seem hard in practice). In §4 we give an *FPT* algorithm for a natural parameterization of the Maximum Agreement Subtree problem having the parameter function 3^k. Many more examples can be found in [**DF98**]. The improvement of parameter functions for *FPT* problems seems to be a productive area for research, where many different ideas and techniques can be employed.

The point of view that parameterized complexity adopts can be summarized in a metaphorical picture. There is an assumption that most interesting problems are hard, so we can picture them as stones, or perhaps planets. The trick is to identify and develop thin zones of computational viability, as suggested in Figure 1.

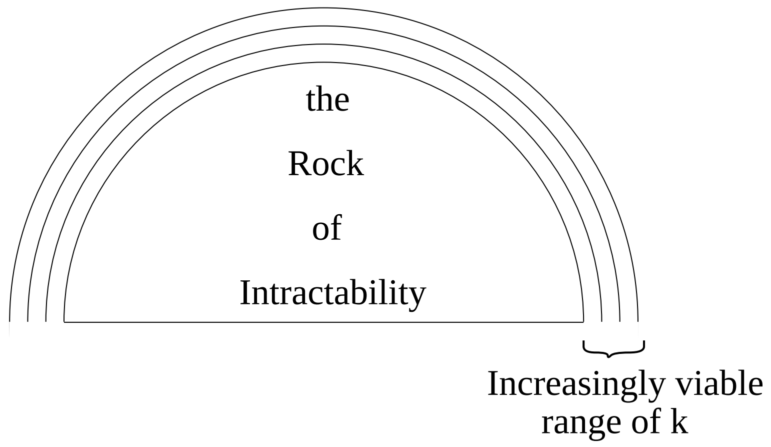

FIGURE 1. The point of view of parameterized complexity is *lichenism.*

3. A Review of the Major Coping Strategies

In §1 we noted the fundamental problem for concrete computational complexity that has emerged in the first decades of computer science:

The need to deal in some systematic way with the pervasive phenomena of computational intractability.

We also noted that it is possible to point to five general *mathematical* strategies that have been proposed so far: (1) average-case analysis, (2) approximation, (3) randomization, (4) fundamentally new kinds of computing devices, and (5) parameterization, as well as another, quasi-mathematical coping strategy, (6) heuristics.

In this section, we review the accomplishments and prospects of these programs.

Average-Case Analysis. In many applications practitioners would be happy with algorithms having good average-case performance. This criterion is implicit in the common practice of evaluating heuristic algorithms on sets of representative benchmarks. The idea that average-case analysis is more realistic than worst-case analysis has been around since the beginnings of theoretical computer science, and its potential role as a method of coping with intractability is discussed by Garey and Johnson in their chapter on this subject in [**GJ79**]. Classic examples of such analysis include the results of Grimmet and McDiarmid [**GM75**] who described a simple graph coloring algorithm with an average performance ratio of 2 for the uniform distribution (where all n vertex graphs are equally likely), and the surprising theorem of Wilf that the average number of steps required by the straightforward backtracking algorithm to determine whether a graph is 3-colorable is 197 for the uniform distribution [**Wilf85**]. [1]

Obtaining theorems concerning average-case complexity is usually mathematically challenging, even for the simplest of algorithms and distributions. As a theoretical program for coping with intractability, average-case analysis seems to be too difficult to carry out for typical hard problems, realistic distributions and sophisticated algorithms. It is also frequently unclear what constitutes a reasonable assumption about the distribution of problem instances, apart from the analytical difficulties of the program.

A completeness notion for average-case complexity has been introduced by Levin [**Lev86**]. This also seems to be difficult to apply, and has only been demonstrated for a few problems. The main weaknesses of average-case analysis as a mathematical program for coping with intractability seem to be:

- In general, it seems to be too difficult to prove the mathematical results that the program calls for.
- The positive toolkit has not developed well. Few general methods are known for designing algorithms with good average-case performance. (One interesting example of a general method has been described by Gustedt and Steger [**GS94**].)

The real strength of average-case analysis as a means of coping is that for most applications of computing it is the *right idea* for how complexity should (usually) be measured, and it is what practitioners generally continue to *do* about complexity measurement in practice, although informally.

Approximation. Perhaps the greatest hopes for a general program for coping with intractability have been pinned on approximation (and also on the average-case analysis of approximation heuristics). Most of the discussion in the chapter on coping with *NP*-completeness in the famous book by Garey and Johnson [**GJ79**] is devoted to explaining the basic ideas of *polynomial time approximation algorithms and schemes.*

Early on, important polynomial time approximation schemes were found for *NP*-complete problems such as Bin Packing [**JDUGG74**] and Knapsack [**IK75**]. However, apart from a few similar results on problems mostly of this same general flavor, it now seems to be clear, on the basis of powerful new proof techniques [**ALMSS92**], that these results are *not* typical for *NP*-hard and otherwise intractable problems. After a long period where the complexity of approximation for most problems remained mysterious, it now seems to be the case that the vast majority of natural *NP*-hard optimization problems probably do not admit efficient approximation schemes.

The study of *the extent to which problems can be approximated* has emerged as a mathematically very rich and productive area of investigation.

[1]For the uniform distribution, it is usually easy to discover that the answer is "no". In fact, what Wilf has shown is that it is usually not even necessary to look at the entire graph!

As pointed out by Hochbaum [**Hoch97**], there is now such an accumulation of interesting and deep results and methods that it is tempting to assess the true difficulty of a hard problem by the degree to which optimal solutions can be approximated in polynomial time. For example, VERTEX COVER (finding a vertex cover that is as small as possible) can be approximated to within a factor of $c = 2$ in polynomial time, and the constant c cannot be improved to $c \leq 16/15$ unless $P = NP$ [**BGS95**]. The contrasts with CLIQUE (find a clique that is as large as possible), which cannot be approximated to better than a factor of $c = n^{0.5-\epsilon}$ for n-vertex graphs, without unlikely complexity-theoretic consequences [**Has96**].

While approximation allows for the clever deployment of mathematics and can be a very effective cure for worst-case intractability when it can be applied, it seems also fair to say that, as with *P versus NP*, most of the results are negative. The main weakness of the program is:

- Most unrestricted classically hard problems cannot be approximated very well.

Randomized Polynomial Time. Randomization is discussed in Chapter 6 of Garey and Johnson [**GJ79**] as a means of avoiding one of the weaknesses of average-case analysis as a coping strategy — the need to have some knowledge in advance of a realistic distribution of problem instances. An algorithm that flips coins as it works may be able to conform to whatever distribution it is given, and either produce an answer in polynomial-time that is correct with high probability (Monte Carlo randomization), or give an answer that is guaranteed to be correct after what is quite likely to be a polynomial amount of time (Las Vegas randomization).

These seemed at first to be potentially very powerful generalizations of polynomial time. Randomized Monte Carlo and Las Vegas algorithms are a workhorse of cryptography [**SS77, GM84**], and have important applications in computational geometry [**Cl87**], pattern matching, on-line algorithms and computer algebra (see [**Karp86**] and [**MR95a**] for surveys), in part because they are often simple to program. Approximate counting is another area of notable success. Randomization is an important new idea that is now applied in many different kinds of algorithms, including approximations and heuristics.

Despite these successes, it now seems that randomized polynomial time is better at delivering good algorithms for difficult problems that "probably" are in P to begin with, than at providing a general means for dealing with intractable problems. There have recently been a number of important results replacing fast probabilistic algorithms with ordinary polynomial time algorithms through the use of sophisticated derandomization techniques [**Rag88, MR95b**]. The main weaknesses of randomization (in the sense of algorithms with performance guarantees) as a general program for coping with intractability are:

- With a few exceptions, it does not seem that randomized polynomial time algorithms are any more effective against problems that are truly hard than ordinary polynomial time algorithms.
- Although the positive toolkit of methods for designing and analyzing randomized algorithms is rich, there is no specifically corresponding negative toolkit that can be used in tandem to negotiate problem complexity and guide the design of effective algorithms.

New Forms of Computation: DNA and Quantum Mechanics. Although these programs have been launched with great fanfare, they so far offer much less of substance than the other items on this list in terms of a general mathematically-powered program for coping with intractability. So far, DNA computing essentially amounts to computation by molecular brute force. Combinatorial explosion can quickly force one to contemplate a very large test tube for brute force computations, despite the fact that information can be stored in molecules with a factor of 10^{12} improved efficiency compared to magnetic tape storage. Mathematically, the program seems to come down to the potential for significant constant speed-ups by means of this physical miniaturization of computing.

It is still unclear whether quantum computers useful for any kind of computation can actually be built. The notion of quantum polynomial time is mathematically interesting, but so far appears to be applicable only to a few special kinds of problems.

The main weakness of these approaches are:

- Practical implementation of the new computing devices seems to be far in the future.
- Biomolecular computing is essentially a physical attack on intractability, not a mathematical one.
- It is unclear whether quantum polynomial time is a significant generalization of ordinary polynomial time, except for a few special kinds of problems.

Parameterization. We can trace the idea of coping with intractability through parameterization to early discussions in Garey and Johnson [**GJ79**], particularly Chapter 4, where it is pointed out that parameters associated with different parts of the input to a problem can interact in a wide variety of ways in producing non-polynomial complexity. The internal structure of an intractable problem — the identification of those aspects (parameters) to which the non-polynomial complexity can be confined — is precisely what is at issue in parameterized complexity.

The structure of non-polynomial complexity is addressed again in Chapter 6 of [**GJ79**], in the discussion of efforts to develop exponential algorithms that improve significantly on simple exhaustive search. A classic example is the algorithm of Nešetřil and Poljak that uses fast matrix multiplication to solve the k-CLIQUE problem for n-vertex graphs in time $O(n^{ck})$ where $c \approx 0.792$ [**NP85**].

A weakness of the parameterized complexity program is that some of the most general and spectacular positive methods, such as the celebrated results of Robertson and Seymour [**RS85**], yield algorithms having parameter functions that are supremely impractical (e.g., towers of 2's of height described by towers of 2's ...). If *tractability* has friends like these, who needs enemies?

The main strengths of parameterization as a program are that it does seem to be very generally applicable to hard problems throughout the classical hierarchy of intractable classes, and it supports a rich toolkit of both positive and negative techniques. The crucial strike against the program seems to be:

- The extent to which *FPT* is really useful is unclear.

Heuristics. Since heuristic algorithms that work well in practice are now, and have always been, the workhorses of industrial computing, there is no question about the ultimate significance of this program for dealing with intractability. There has recently been a revival of interest in obtaining systematic empirical performance evaluations of heuristic algorithms for hard problems [**BGKRS95, Hoo95, JM93, JT96**]. There have been vigorous developments of new ideas for designing heuristic algorithms, particularly new ideas employing randomization in various ways. These approaches frequently have colorful and extravagant names based on far-fetched analogies in other sciences, such as *simulated annealing*, *genetic algorithms*, *cellular automata*, *neural nets*, *great deluge algorithms* [**Due93**], and *roaming ants* [**CDM92**]. Many of these can be considered as variants on the basic technique of local search.

The main problem with considering heuristics as a systematic program for coping with intractability is that it is not a coherent mathematical program. The positive toolkit properly includes voodoo and the kitchen sink. As a program, it doesn't call for any theorems, only empirical performance. The undeniable successes of sometimes "mindless" and generally unanalyzable heuristic algorithms puts computer science theory in an uncomfortable position.

Heuristics based on local search perhaps come the closest to constituting a mathematically articulated general coping strategy for intractability (see the articles in [**AL97**]). There is a negative toolkit (of sorts) based on the notion of *polynomial local search (PLS)* problems and *PLS*-completeness, introduced by Johnson, Papadimitriou and Yannakakis [**JPY88**]. Although a number of local search problems have been shown to be complete, the reductions are quite demanding (so there aren't very many such results), and there is furthermore a notable peculiarity of this framework. For a concrete example, because the TRAVELING SALESMAN PROBLEM is *NP*-hard, one resorts to a local search heuristic based on the k-Opt neighborhood structure, such as the Lin-Kernighan algorithm [**LK73**], and this is considered to be a particularly successful local search heuristic. Yet, Krentel has shown that for $k = 8$, this neighborhood structure is *PLS*-complete [**Kr90, JMc97**]. This seems like a weapon firing in the wrong direction, or perhaps just a kind of reiteration that TSP is a hard problem. It is unclear how *PLS*-completeness provides any guidance in designing local search heuristics.

The main difficulty is summarized:

- Although heuristics are the principal coin of the realm in practical computing, the design of heuristics is not well-organized as a mathematical program.

Some general remarks. If these research programs were the *Knights of Theoretical Computer Science* who would deal with the *Dragon of Intractability*, we would seem to have: one that is proceeding on foot and armed with a cudgel (actually just a hired ruffian), one in mystic battle-wear and armaments riding backwards away from the fray, one equipped with spells and lucky charms effective against small lizards, one who is riding in a baby carriage armed with a rattle, and one who is on horse, going in the right direction, and armed with a lance — but a lance that is effective only after stabbing the Dragon a number of times bounded by

$$2^{2^{2^{2^{2^{2^{2^{k}}}}}}}$$

where k is ... unfortunately, it doesn't matter.

Leaving fanciful impressions aside, it seems fair to say that the central problems of theoretical computer science, both structural and concrete, have turned out to be *much* harder to crack than was hoped in the early years of the field.

Mathematical computer scientists have so far had little to offer of any general value from a systematic perspective to practitioners who are frequently concerned with very specific hard problems (such as STRIPS PLANNING [**FN71**]). What seems to be happening is that as mathematical science makes no progress on systematically addressing intractability, we are visited by a series of fads based on turning to *some other science* as a source of solutions, often accompanied by a fanfare of "paradigm shifts" such as:

- Turning to Physics for metaphors, and getting algorithms called "simulated annealing".
- Turning to Brain Science for metaphors, and getting "neural nets".
- Turning to Evolutionary Biology for metaphors, and getting "genetic algorithms".
- Turning to Physics for more metaphors, this time about "phase transitions".

Each of these essentially shallow notions has in turn been widely and enthusiastically embraced by practitioners who must deal with their favorite hard problems somehow. From the point of view of computer science theory, each of these algorithmic fads is disappointing — they provide little opportunity to deploy the power of mathematical reasoning. It is frequently unclear if their effectiveness is much more than superstition. (If this summary of the field seems unpleasantly harsh, then look again at papers such as [**DKL96**].)

The present situation of theoretical computer science is that it has *not* been very successful in coming up with a viable systematic program for dealing in a general way with computational intractability in its own native terms (i.e., in terms of productive and interesting mathematical science). No wonder that practitioners turn in other directions, and complain!

4. Industrial Strength *FPT*

There are two main points that we will argue in this section:

1. The notion of *FPT*, in many cases, simply provides a new name and a new perspective on heuristic algorithms already in use. *FPT* algorithms frequently turn out to be *what clever practitioners implemented* after their problems were proved *NP*-hard. Where natural instance distributions exhibit limited parameter ranges, these have often been implicitly exploited in the design of useful heuristics.
2. The parameterized complexity perspective can lead to useful algorithms in several different ways, including:
    - Directly and analytically, when the parameter function is reasonable (i.e., not too fast-growing) and the parameter describes a restriction of the general problem that is still useful.
    - Directly and empirically, in cases where the analytic bound on the running time of the *FPT* algorithm turns out to be too crude or pessimistic, that is, where the algorithm turns out to be useful for larger parameter ranges than the parameter function would indicate.
    - By supplying guidance in the systematic design of heuristic algorithms in the form of explicit general methods based on *FPT* techniques, using the theory to understand "the internal structure of the complexity of the problem" by identifying those parameterizations of the problem that are *FPT* and those that are probably not (because they are hard for $W[1]$, the parameterized analog of *NP*).
    - Via methodological connections between *FPT* and the design of efficient polynomial-time approximation schemes (where the relevant implicit parameter is $k = 1/\epsilon$, for approximations to within a factor of $(1+\epsilon)$ of optimal) and other approximation heuristics.

4.1. Various *FPT* Algorithms and Heuristics for Vertex Cover.

We can continue to use the Vertex Cover problem to illustrate many of the main ideas. This is a useful example because it is a simple problem to describe, and because a wide variety of *FPT* algorithmic techniques can be applied to it. We will see how these different techniques can be systematically adapted into a corresponding variety of heuristics.

We begin by describing a new *FPT* algorithm for VERTEX COVER that is currently the best known. It is based on two simple, but standard methods, *reduction to a problem kernel*, and *search trees*. More examples and discussion of these methods can be found in [**DF95c, DF98, KST94, MR98**].

ALGORITHM 4.1. (An Improved Direct *FPT* Algorithm for VERTEX COVER.) The algorithm proceeds in two phases. In the first phase we compute the *kernel* of the given instance (G,k) or answer "no". The kernel is an instance (G',k') where $|G'| \leq k^2$ and $k' \leq k$ such that G' has a vertex cover of size k' if and only if G has a vertex cover of size k. The reduction from (G,k) to (G',k') is computable in time $O(kn)$, where k is the number of vertices in G. (That is, this is a *polynomial time* parametric transformation of VERTEX COVER *to itself*, such that the target instance has size bounded by a function of the parameter k.) Given that we can do this, we have immediately demonstrated membership in *FPT* for VERTEX COVER, since the question for (G',k') can now be answered in time bounded by a function of k simply by an exhaustive analysis of G'.

Phase 1 (Reduction to a Problem Kernel): Starting with (G,k) we apply the following reduction rules until no further applications are possible (the justifications for the reductions are given below):

(0): If G has a vertex v of degree greater than k, then replace (G,k) with $(G-v,k-1)$.

(1): If G has two nonadjacent vertices u,v such that $|N(u) \cup N(v)| > k$, then replace (G,k) with $(G+uv,k)$.

(2): If G has adjacent vertices u and v such that $N(v) \subseteq N[u]$, then replace (G,k) with $(G-u,k-1)$.

(3): If G has a pendant edge uv with u having degree 1, then replace (G,k) with $(G-\{u,v\},k-1)$.

(4): If G has a vertex x of degree 2, with neighbors a and b, and none of the above cases applies (and thus a and b are not adjacent), then replace (G,k) with (G',k) where G' is obtained from G by:
- Deleting the vertex x.
- Adding the edge ab.
- Adding all possible edges between $\{a,b\}$ and $N(a) \cup N(b)$.

(5): If G has a vertex x of degree 3, with neighbors a,b,c, and none of the above cases applies, then replace (G,k) with (G',k) according to one of the following cases depending on the number of edges between a, b and c.

(5.1): There are no edges between the vertices a,b,c. In this case G' is obtained from G by:
- vertices v_2v_3. Deleting vertex x from G.
- Adding edges from c to all the vertices in $N(a)$.
- Adding edges from a to all the vertices in $N(b)$.

- replacement be seen Adding edges from b to all the vertices in $N(c)$.
- Adding edges ab and bc.

(5.2): There is exactly one edge in G between the vertices a, b, c, which we assume to be the edge ab. In this case G' is obtained from G by
- Deleting vertex x from G.
- Adding edges from c to all the vertices in $N(b) \cup N(a)$.
- Adding edges from a to all the vertices in $N(c)$.
- Adding edges from b to all the vertices in $N(c)$.
- Adding edge bc.
- Adding edge ac.

The reduction rules described above are justified as follows:

(0): Any k-element vertex cover in G must contain v, since otherwise it would be forced to contain $N(v)$, which is impossible.

(1): It is impossible for a k-element vertex cover of G not to contain at least one of u, v, since otherwise it would be forced to contain all of the vertices of $N(u) \cup N(v)$.

(2): If a vertex cover C did not contain u then it would be forced to contain $N[v]$. But then there would be no harm in exchanging v for u.

(3): If G has a k-element vertex cover C that does not contain v, then it must contain u. But then $C - u + v$ is also a k-element vertex cover. Thus G has a k-element vertex cover if and only if it has one that contains v.

(4): We first argue that if G has a k-element vertex cover C, then it must have one with one of the following forms:
1. C contains a and b, or
2. C contains x but neither of a, b, and therefore also contains $N(a) \cup N(b)$.

If C did not have either of these forms, then it must contain exactly one of a, b, and therefore also x. But in this C can be modified to form 1. If G has a k-element vertex cover of the form 1, then this also forms a k-element vertex cover in G'. If G has a k-element vertex cover of the form 2, then this same set of vertices with x replaced by either a or b is a k-element vertex cover in G'. Conversely, suppose G' has a k-element vertex cover C. If C contains both a and b, then it is also a k-element vertex cover in G. Otherwise, it must contain at least one of a, b, suppose a. But then the edges from b to all of the vertices of $N(a) \cup N(b)$ in G' force C to contain $N(a) \cup N(b)$. So $C - a + x$ is a k-element vertex cover in G.

(5.1): Let C denote a k-element vertex cover in G. If C does not contain x, then necessarily C contains $\{a, b, c\}$. In this case C is also a k-element vertex cover in G'. Assume that C contains x. We can assume (easy to check) that at most one of the vertices of $\{a, b, c\}$ belongs to C. If none of the vertices of $\{a, b, c\}$ belongs to C, then

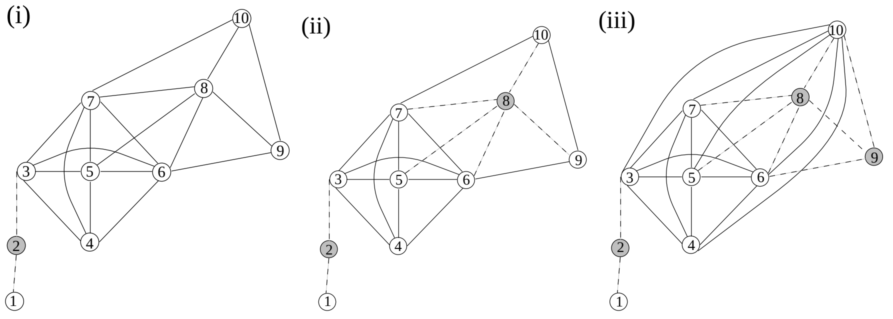

FIGURE 2. Let $k = 8$. (i) In the given graph $(1, 2)$ is a pendant edge (case (3)). $k' = 7$, $G' = G - \{1, 2\}$. 2 is added to the vertex cover. (ii) Vertices 8 and 10 are adjacent and $N(10) \subseteq N[8]$ (case (2)). (iii) The degree of vertex 9 is 2 (case (4)). $k''' = 7$, G''' is received by adding the following edges to $G'' - \{9\}$: $(10, 3)$, $(10, 4)$, $(10, 5)$, $(10, 6)$.

either b or c belongs to the vertex cover of G'. If $a \in C$ then $C - x + c$ is a vertex cover for G', if $b \in C$ then $C - x + a$ is a vertex cover for G', and if $c \in C$ then $C - x + a$ is a vertex cover for G'.

(5.2): Let C denote a k-element vertex cover in G. If C does not contain x, then necessarily C contains $\{a, b, c\}$. In this case C is also a k-element vertex cover in G'. Assume that C contains x. We can assume (easy to check) that exactly one of the vertices of $\{a, b\}$ belongs to C. W.l.o.g we assume $a \in C$. Then $C - x + c$ is a vertex cover for G', if $b \in C$ then $C - x + c$ is a vertex cover for G'.

It is easy to see that at the end of Phase 1 we have reduced (G, k) to (G', k') where G' has minimum degree 4, if we have not already answered the question. Furthermore, simply because of the reduction rules (1) and (2) we can conclude that the answer is "no" if the number of vertices in G' is more than k^2. Phase 1 is a good example of what is meant by the *FPT* technique of *reduction to a problem kernel.* If we still have no answer about the original input (G, k), then we are left with considering (G', k') where $|G'| \leq k^2$ and $k' \leq k$. A demonstration that the problem is in *FPT* is now trivial, since we can just (in the absence of further ideas) *exhaustively* answer the question for the kernel (G', k'). However, we can do a little better by analyzing the kernel instance by means of a search tree.

Figure 2 shows an example of the kernelization procedure of Phase 1.

Phase 2 (Search Tree): In this phase of the algorithm we build a search tree of height at most k. The root of the tree is labeled with the output (G', k') of Phase 1. We will describe various rules for deriving the children of a node in the search tree. For example, we can note that for a vertex v in G', and for any vertex cover C, either $v \in C$ or $N(v) \subseteq C$. Consequently we could create two children, one labeled with $(G' - v, k' - 1)$, and the other labeled with $(G' - N[v], k' - \deg(v))$. In our algorithm, we perform this branching if there is a vertex of degree at least 6. By repeating this branching procedure, at each step reapplying the reductions of Phase 1, we can assume that that at each leaf of the resulting search tree we are left with considering a graph where every vertex has degree 4 or 5. If there is a vertex x of degree 4, then the following branching rules are applied. Suppose that the neighbors of a vertex x are $\{a, b, c, d\}$. We consider various cases according to the number of edges present between the vertices a, b, c, d.

Note that if not all of $\{a, b, c, d\}$ are in a vertex cover, then we can assume that at most two of them are.

Case 1. The subgraph induced by the vertices a, b, c, d has an edge, say ab.

Then c and d together cannot be in a vertex cover unless all four of a, b, c, d are there. We can conclude that one of the following is necessarily a subset of the vertex cover C and branch accordingly:

1. $\{a, b, c, d\} \subseteq C$
2. $N(c) \subseteq C$
3. $\{c\} \cup N(d) \subseteq C$.

Case 2. The subgraph induced by the vertices a, b, c, d is empty. We consider three subcases.

Subcase 2.1 Three of the vertices (say a, b, c) have a common neighbor y other than x.

Then when not all of a, b, c, d are in a vertex cover, x and y must be. We can conclude that one of the following is a subset of the vertex cover C and branch accordingly:

1. $\{a, b, c, d\} \subseteq C$
2. $\{x, y\} \subseteq C$.

Subcase 2.2 If Subcase 2.1 does not hold, then there may be a pair of vertices who have a total of six neighbors other than x, suppose a and b. If all of a, b, c, d are not in the vertex cover C then $c \notin C$, or $c \in C$ and $d \notin C$, or both $c \in C$ and $d \in C$ (in which case $a \notin C$ and $b \notin C$). We can conclude that one of the following is a subset of the vertex cover C and branch accordingly:

1. $\{a, b, c, d\} \subseteq C$
2. $N(c) \subseteq C$
3. $\{c\} \cup N(d) \subseteq C$
4. $\{c, d\} \cup N(\{a, b\}) \subseteq C$.

Subcase 2.3 If Subcases 2.1 and 2.2 do not hold, then the graph must have the following structure in the vicinity of x: (1) x has four neighbors a, b, c, d and each of these has degree four. (2) There is a set E of six vertices such that each vertex in E is adjacent to exactly two vertices in $\{a, b, c, d\}$, and the subgraph induced by $E \cup \{a, b, c, d\}$ is a subdivided K_4 with each edge subdivided once. In this case we can branch according to:

1. $\{a, b, c, d\} \subseteq C$
2. $(E \cup \{x\}) \subseteq C$.

If the graph G is regular of degree 5 (that is, there are no vertices of degree 4 to apply one of the above branching rules to) and none of the reduction rules of Phase 1 can be applied, then we choose a vertex x of degree 5 and do the following. First, we branch from (G, k) to $(G - x, k - 1)$ and $(G - N[x], k - 5)$. Then we choose a vertex u of degree 4 in $G - x$ and branch according to one of the above cases. The net result of these two combined steps is that from (G, k) we have created a subtree where one of the following cases holds:

1. There are four children with parameter values $k - 5$, from Case 1.
2. There are three children with parameter values $k_1 = k - 5$, $k_2 = k - 5$ and $k_3 = k - 3$, from Subcase 2.1.
3. There are five children with parameter values $k_1 = k - 5$, $k_2 = k - 5$, $k_3 = k - 5$, $k_4 = k - 6$ and $k_5 = k - 9$, from Subcase 2.2.

Note that if reduction rule (2) of Phase 1 cannot be applied to $G - x$, then at least one of the neighbors of u has degree 5, and so Subcase 2.3 is impossible.

The bottleneck recurrence comes from the degree 5 situation which produces four children with parameter values $k - 5$. The total running time of the algorithm is therefore $O(r^k k^2 + kn)$, where $r = 4^{1/5}$, or $r = 1.31951$ approximately. This time bound is a slight improvement on the $r = 1.32472$ of [**BFR98**]. (The tiny difference amounts to a 21% improvement in the running time for $k = 60$.)

The parameter function of our analysis of the running time of Algorithm 4.1 indicates that this "exact" algorithm is useful for input graphs of any size, so long as the parameter k is no more than about 60. There are a number of uses of VERTEX COVER in analyzing biological sequence data, and for this reason, essentially the above algorithm has been implemented as part of the DARWIN project at ETH [**HGS98**]. It turns out that the useful parameter range of $k \leq 60$ indicated by the parameter function above is overly pessimistic. In practice, the algorithm seems to be useful for k up to around 200. Note that if you run this algorithm on (G, k) where k is apparently too large, the worst that can happen is that the algorithm fails to finish in a reasonable amount of time. If it terminates, then it does give the correct answer.

The two phases of the above algorithm are independent. If one intended to solve the general VERTEX COVER problem by simulated annealing (or any other method), it would still make sense to apply Phase 1 *before* doing the simulated annealing, since the "simplifications" accomplished by the kernelization require only polynomial time (that is, they in no way depend on k being small). Consequently, Phase 1 is a reasonable first step for *any* general algorithmic attack on the *NP*-complete VERTEX COVER problem. We can codify this discussion by describing the following heuristic algorithm for the *general* VERTEX COVER problem. That is, the following algorithm, although it is based on *FPT* methods, has nothing to do with small parameter ranges, and it runs in polynomial time.

ALGORITHM 4.2. (A General Heuristic Algorithm for VERTEX COVER Based on Kernelization.) The algorithm simply reduces the input graph G to nothing by repeating the following two steps:

1. Apply Phase 1 of Algorithm 4.1.
2. Choose a vertex of maximum degree.

The reduction path gives an approximate minimum vertex cover for G. To see that this works correctly, it is necessary to observe that for each of the rules of Phase 1 reducing (G, k) to (G', k'), and given any vertex cover C' in G', we can reverse the reduction to obtain a vertex cover C in G.

In Algorithm 4.2, we have simply replaced Phase 2 of Algorithm 4.1, which is exponential in k, with a well-known approximation heuristic for this problem. However, we could also *adapt* Phase 2 by *simply not exploring all of the branches of the search tree*, either by making a random selection, or by branch selection heuristics, etc. The following is a natural adaptation of Algorithm 4.1 that exploits this idea for designing heuristics based on the *FPT* search tree technique.

ALGORITHM 4.3. (A General Heuristic Algorithm for VERTEX COVER Based on Kernelization and a Truncated Search Tree.) This is the same as Algorithm 4.1, except that in the second phase of building the the search tree, we do two things. First of all, we decide in advance how large of a search tree we can afford, and we build the tree until we have this number of leaves. Then for each leaf of the search tree, we apply an approximation heuristic such as the greedy method used in Algorithm 4.2, and take the best of all the solutions computed from the leaf instances of the search tree.

The search tree is developed according to the following branching heuristic. Given the instance (G, k) as the label on a node of the search tree to be expanded, we calculate the most valuable branch in the following way:

1. Find a vertex a of maximum degree r in G.
2. Find a pair of vertices b, c such that $s = |S|$, where $S = N(b) \cap N(c)$ is maximized.
3. Determine which of the two equations $x^r - x^{r-1} - 1 = 0$ and $x^s - x^{s-2} - 1 = 0$ has the smallest positive real root. If the first equation, then branch to $(G - a, k - 1)$ and $(G - N[a], k - r)$. If the second, then branch to $(G - b - c, k - 2)$ and $(G - S, k - s)$. (This information can be precomputed and determined by table lookup.)

The branching heuristic above is justified because it essentially chooses the branching rule to apply according to the heuristic criterion that if it were the only rule applied in building a tree, then this would result in the smaller tree. (This could obviously be generalized by considering 3-element subsets of the vertices, or by employing a different criterion for choosing a branching rule.)

Algorithm 4.3 is illustrated in Figure 3.

In view of the remarkably nice *FPT* algorithms we now have for VERTEX COVER it is both ironic and instructive that the VERTEX COVER problem

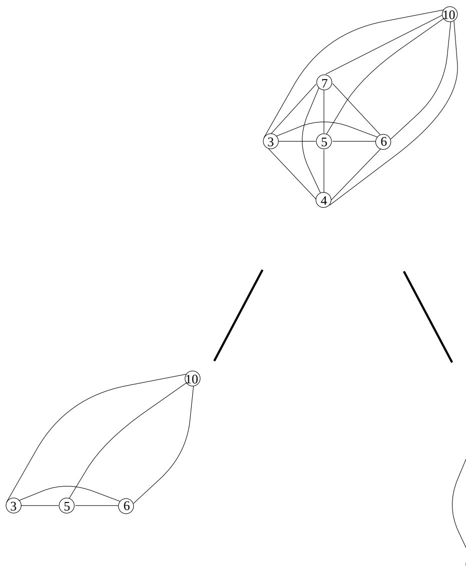

FIGURE 3. The resulting graph in Figure 1 is taken as root of the search tree in Algorithm 3. Here $a = 3$, $b = 4$, and $c = 7$.

was first proved to be fixed-parameter tractable by the application of methods that are much more complicated than the problem actually requires. In fact, this happened in print at least three times [**FL87, Jo87, PY96**]. The first of these approaches is based on the Robertson-Seymour theorems, the second on finite-state dynamic programming on graphs of bounded treewidth, and the third on maximum matching. Probably the correct moral to draw from this history is that the design of *FPT* algorithms has its own distinctive issues and opportunities, and is not necessarily well-served by the habits we have developed for the design of polynomial time algorithms.

With respect to the connection between *FPT* and practical heuristics, the correct point of view seems to be that *all of the various FPT methods that can be applied to a problem may be useful in the design of heuristics.* We next describe a new, utterly *inefficient* (in terms of the parameter function) *FPT* algorithm for VERTEX COVER, based on completely different *FPT* techniques, and then describe how these methods can contribute, nevertheless, to the design of an interesting and apparently powerful heuristic.

DEFINITION 4.1. $G \geq H$ if there is a subgraph $G' \subseteq G$ and a surjective map $h : V(G') \rightarrow V(H)$ with the property that for all $uv \in E(H)$, there are vertices $x, y \in V(G)$ such that $h(x) = u$, $h(y) = v$ and $xy \in E(G)$. (It is easy to verify that this defines a partial order on graphs.)

If h is a function that k-colors the vertices of $G = (V, E)$, $h : V \rightarrow \{1, ..., k\}$, then we will write $h(G)$ to denote the graph with the vertex set

$h(V) \subseteq \{1, ..., k\}$ and the edge set

$$\{ij : \exists x, y \in V \text{ with } h(x) = i, h(y) = j \text{ and } xy \in E\}.$$

LEMMA 4.2. *For each fixed k there is a finite set $\mathcal{O}_k$ of graphs, such that for every graph G, G has a k-element vertex cover if and only if $\forall H \in \mathcal{O}_k : G \not\geq H$.*

PROOF. By the Graph Minor Theorem [**RS96**], it is enough to argue that if H is a minor of G then $G \geq H$. But this is essentially trivial, since one way of viewing the fact that H is a minor of G is that there is a *folio representation* of H in G. That is, there is a set of disjoint connected subgraphs $G_v \subseteq G$, one such subgraph for each vertex $v \in V(H)$, such that if $uv \in E(H)$ then there are vertices $x \in G_u$ and $y \in G_v$ with $xy \in E(G)$. From this it is easy to construct a partial function $h : V(G) \to V(H)$ showing that $G \geq H$. □

The celebrated Graph Minor Theorem used in the above argument simply states that every set of finite graphs has a finite number of minimal elements in the partial ordering of graphs by minors. For a gentle survey of this result with an eye to algorithmic applications, see [**FL88**].

LEMMA 4.3. *For every fixed graph H, it can be determined in time $O(n \log n)$, for an input graph G on n vertices, whether $G \geq H$.*

PROOF. Note that we are essentially proving that the problem of determining whether $G \geq H$ is in *FPT* for the natural parameter H. As a consequence of a theorem of Mader [**Mad72**], there is a constant c_H such that if a graph G on n vertices has more then $c_H n$ edges then necessarily G has H as a minor and therefore $G \geq H$. In linear time we can determine if this provides a reason to answer "yes" and otherwise we can assume a linear bound on the number of edges of G.

We use the powerful and general *FPT* method of *color-coding* introduced by Alon, Yuster and Zwick. In [**AYZ94**] it is shown that for every k there is a family $\mathcal{H}_k$ of functions $h : \{1, ..., n\} \to \{1, ..., k\}$ with the property that if S is any subset of $\{1, ..., n\}$ of size k, then $\exists h \in \mathcal{H}_k$ such that h maps S 1:1 onto $\{1, ..., k\}$. The crucial thing is that the family of functions $\mathcal{H}_k$ is not very large. In [**AYZ94**] it is shown that it is possible to produce such families of hash functions with $|\mathcal{H}_k| \leq 2^{O(k)} \log n$. (Note the reasonable parametric function, in the sense of the discussion in §2.)

Suppose H has m edges. Then $G \geq H$ if and only if there is a set of at most $2m$ vertices of G that can be mapped onto the vertices of H in order to witness this fact.

Let $\mathcal{F}_{2m}$ denote the set of all functions from the set $\{1, ..., 2m\}$ onto the vertex set of H. Then $G \geq H$ if this can be detected by the composition $f = g \circ h$ of some function $h \in \mathcal{H}_{2m}$ and $g \in \mathcal{F}_{2m}$, by having H isomorphic to $f(G)$. Our algorithm thus consists simply of computing $f(G)$ for all such functions f. This can clearly be accomplished in time $O(n \log n)$ with a hidden constant that is exponential in the size of the fixed graph H. □

The two lemmas above allow us to conclude immediately that Vertex Cover is in *nonuniform FPT*. Nonuniform, because so far we can only conclude that for each fixed k there is a different algorithm for k-Vertex Cover, based in each case on the obstruction set $\mathcal{O}_k$, which is different for each k, and not in any obvious way computable. In order to combine these separate algorithms into a single (uniform) *FPT* algorithm, we will use the self-reduction technique of [**FL94**]. The subroutine that we need is described by the following lemma.

Lemma 4.4. *Suppose A is an oracle (black box) for the* Vertex Cover *decision problem. Then by making $O(k^2)$ calls to the oracle, we can compute a k-element vertex cover for a graph G on n vertices (if it has one).*

Proof. First note that by attaching pendant edges to vertices of G, we immediately have an easy algorithm that makes at most n calls to the oracle by repeatedly "probing" the graph by attaching a pendant edge to the vertices of G. Note that if uv is a pendant edge where u has degree 1, then if G has any k-element vertex cover, then it has one including the vertex v. We simply keep probing the vertex set until we have attached pendant edges to k vertices.

The operation of *shattering* a vertex x is accomplished by:

- Deleting x.
- Adding pendant edges to each vertex $y \in N(x)$.

If G has a k element vertex cover, then for any partition of the vertex set of G into $k+1$ classes there must be at least one class that is *safe to shatter* in the sense that if G' is obtained from G by shattering every vertex in the class, then G' has a k-element vertex cover that is also a vertex cover for G. Thus by making $k+1$ calls to the oracle, one for each such G', we can discover at least one new vertex that can be tagged with a pendant edge. At an intermediate step of the algorithm, there is some set S of $k' < k$ vertices of G that are tagged with pendant edges, and we know these k' vertices belong to some k-element vertex cover. We can delete a vertex $z \notin S$ if $N(z) \subseteq S$ (since the vertices of S are forced to be in the vertex cover, it is safe to assume that z is not). The remaining vertices are partitioned into $k - k' + 1$ approximately equal-sized classes. By inquiring about the graphs G' obtained by shattering these classes, we can discover at least one new vertex to add to S. Thus $O(k^2)$ calls to the oracle are sufficient. □

Algorithm 4.4. (A Direct But Completely Impractical *FPT* Algorithm for Vertex Cover Based on Well-Quasiordering, Hashing, and Fast Self-Reduction.) The algorithm receives an input instance (G, k) and begins to build a list of the elements of $\mathcal{O}_k$ by systematically generating *all* finite graphs, and checking each one to see if it should be added to the list (this is easy to determine). Since by what we have argued so far we have no way of knowing when the list is complete, we interleave this process with another, until we have decided whether to answer "yes" or "no". We call this procedure for finding a new obstruction *Process 1.*

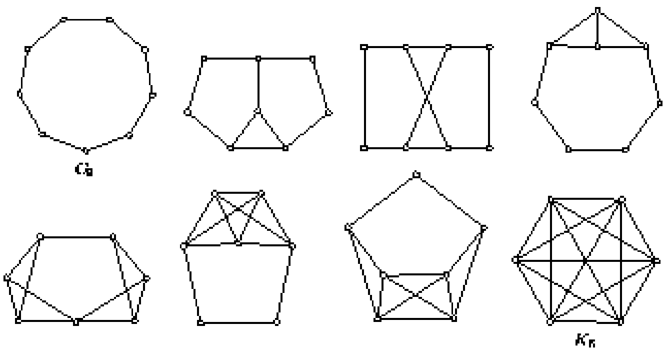

FIGURE 4. The connected obstructions for 4-VERTEX COVER

Process 2 is executed whenever Process 1 has added something new to the list of $\mathcal{O}_k$. Process 2 acts under the *assumption* that the list is complete. If this were so, then by the means described by Lemmas 4.2 and 4.3, the list provides an "oracle" for the decision problem, by checking, for each H on the list, whether $G \geq H$.

Each call to this decision problem "oracle" requires time $O(f(k)n \log n)$, where f is some function of the parameter. We use the oracle to obtain either a definite reason to answer "no", by discovering that $G \geq H$ for some $H \in \mathcal{O}_k$, or a definite reason to answer "yes", by actually identifying and checking a k-element vertex cover in G. In order to obtain the latter evidence, we use the self-reduction algorithm of Lemma 4.4. Either of these efforts to give a definite answer may in fact succeed, even though our assumption that the list is complete is incorrect. (If the assumption is correct, then of course we will succeed in giving a definite answer.)

The only remaining possibility is that something weird happens. The incorrect assumption may cause the self-reduction algorithm to take too long or to output an incorrect solution. These malfunctions are easy to recognize. If we discover that the assumption was incorrect in this manner, then we return to Process 1 until we generate a new graph to add to the list of known elements of $\mathcal{O}_k$. Since $\mathcal{O}_k$ is finite, we alternate between the two processes at most $|\mathcal{O}_k|$ times.

The running time can be calculated to be $O(f(k)n \log^2 n)$ for a function $f(k)$ that grows explosively mainly because of the size of the obstruction sets $\mathcal{O}_k$.

The set of obstructions for $k = 4$ is shown in Figure 4. In principle, the set $\mathcal{O}_k$ can be mechanically computed for any k. The set shown in Figure 4 was computed by Cattell and Dinneen [**CD94**].

Algorithm 4.4 is a nice example of the use of some very powerful and general *FPT* methods, and it is a good example as well of an *FPT* result that is utterly impractical as a direct algorithm. Nevertheless, we can adapt the *methods* of Algorithm 4.4 in designing new and interesting heuristics for VERTEX COVER. General heuristic design strategies that correspond to some of the main *FPT* methods are displayed in Table 2. 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.

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.

TABLE 2. Some *FPT* Methods and Heuristic Strategies

We have the following heuristic algorithm for VERTEX COVER adapted from Algorithm 4.4.

ALGORITHM 4.5. (A New Family of Heuristics for VERTEX COVER.)

The heuristic algorithm that we describe is *self-analyzing* in the sense that it computes both a vertex cover $C \subseteq V$ for an input graph $G = (V, E)$, and a guarantee r that C is within a factor of r of optimal.

Step 1: An Initial Solution. The first step of the algorithm is to compute a vertex cover that is within a factor of 2 of optimal, using one of the well-known approximation algorithms that provide this performance guarantee.

Step 2: Improving the Upper Bound. Knowing that G has a vertex cover of size k_{upper}, we attempt to show that that it has one of size $k' < k_{\text{upper}}$. To do this, we use a strategy suggested by Lemma 4.4. If $S \subseteq V$ is a set of vertices of G, define the *shattering of G with respect to S* to be the graph obtained by "unplugging" any edges incident on vertices of S. If S consists of a single vertex, then this coincides with the definition given in the proof of Lemma 4.4. If two vertices $u, v \in S$ are adjacent, then this results in a K_2 from uv being unplugged from both endpoints. There are two salient facts that we use concerning this operation:

- If (G, k) is a yes-instance, then for any $(k+1)$-partition of the vertex set, there must be at least one class S of the partition such that (G', k) is also a yes-instance, where G' is obtained by shattering G with respect to S.
- If G' is the result of shattering G with respect to S, and C' is a vertex cover in G', then we can easily compute from C' a vertex cover C for G with $|C| \leq |C'|$.

Typically, the result G' of shattering G with respect to S will have many vertices of degree 1, and hence G' can be reduced by applying the rules for the kernelization phase of Algorithm 4.1. In this way, we obtain a much smaller instance from which we can possibly compute a vertex cover for G of the targeted size to improve the upper bound. Using the facts above, we develop a search tree based on some randomly or heuristically chosen partitions, and explore some of branches (where a branch is given by shattering on a class of the partition).

There are various ways to work out the details of this exploration process. We might think of the tree as organized into *lower branches* (closer to the root) and *upper branches*. The lower branches create a search space of affordable size consisting of graphs that are somewhat smaller than G, but a search space that is guaranteed to contain yes-instance under the assumption that G is a yes-instance (because the lower branches include *all* shatterings for some partition). The upper branches are developed by repeating the shattering process enough times, but only exploring some of the branches, until the resulting graphs are completely solved. The exploration of the upper branches could also be based on a greedy heuristic.

Step 3: Improving the Lower Bound. Knowing a lower bound k_{lower} on the minimum size of a vertex cover in G, we attempt to prove a better lower bound $k > k_{\text{lower}}$. To do this, we use a strategy suggested in part by Lemma 4.2. However, before doing anything fancy, we should compute a maximal matching in G to see if there is an easy way to improve the lower bound by detecting the easy obstruction $(k+1)K_2 \in \mathcal{O}_k$.

Note that if (G, k) is a no-instance, then for any $(2k+3)$-coloring of the vertex set, there must be at least one class that can be *pinched* in the following way, to produce a (smaller) graph G' such that (G', k) is also a no-instance. That this is true follows from Lemma 4.2 and the fact that the maximum number of vertices in in $\mathcal{O}_k$ is $2k+2$ (attained uniquely by $(k+1)K_2$). This can be used to develop the lower branches of a search tree as in Step 2 which is similarly guaranteed to have at least one branch to a simpler graph from which a proof that (G, k) is a no-instance can be obtained (if it is a no-instance).

For the upper branches of this search tree we can employ a partial exploration of the obstruction set for $\mathcal{O}_k$ and a partial exploration of the family of hash functions used in the proof of Lemma 4.3 to see if a reason for answering "no" can be obtained. It is easy to generate elements in $\mathcal{O}_k$ by taking disjoint unions of obstructions for smaller parameter values. For example $C_5 \in \mathcal{O}_2$ and $K_6 \in \mathcal{O}_4$ so $(C_5 \cup K_6) \in \mathcal{O}_7$.

Instead of incrementally applying Steps 2 and 3 separately to close the gap between $k_{rmupper}$ and k_{lower}, some form of binary search could be adopted. For example, the first round of such a strategy might be to apply Steps 2 and 3 to $k = (k_{\text{upper}} + k_{\text{lower}})/2$.

The main point in describing the above heuristic (which has not yet been implemeted) is to illustrate how *FPT* techniques can be adapted into heuristic algorithms.

4.2. A Useful Parameterized Algorithm for MAST. We next describe a useful direct *FPT* algorithm for the MAXIMUM AGREEMENT SUBTREE (MAST) problem defined in §1, when it is restricted to binary trees, a reasonable restriction for biological applications. Apart from the intrinsic interest of this result, it is a nice example of two important points concerning *FPT* algorithms.

- Our algorithm for MAST uses an algorithm for VERTEX COVER as a subroutine. Useful *FPT* algorithms lead to other useful *FPT* algorithms, as one might naturally expect.
- There is already a polynomial time algorithm for MAST for binary trees, so why bother with an exponential *FPT* algorithm? The answer is that the polynomial time algorithm for MAST due to [**FPT95**] runs in time $O(rn^3)$ for r trees on a set of n species. The algorithm we describe requires time $O(c^k rn \log n)$ where c is a constant less than 3. Consequently, this is an example of a situation *where a classically exponential FPT algorithm may be preferable to a polynomial time algorithm.*

THEOREM 4.5. *The parameterized MAST problem can be solved in time $O(c^k rn \log n)$ for r binary trees on n species.*

SKETCH. The input to the problem is a set of rooted binary trees $T_1, ..., T_r$ each having n leaves labeled in 1:1 correspondence with a set X of n species. The problem is to determine if it is possible to delete at most k species from X to obtain a set X' on which all of the trees *agree*. In considering this problem, there is an elegant point of view developed by Bryant [**Bry97**] based on *triples*. If $\{a, b, c\}$ is a set of three species in X, then the restriction of each of the trees T_i to these three species must be one of the three possible alternatives (using parenthetical notation to represent trees): (a,(b,c)), (b,(a,c)), or (c,(a,b)). If two or more of these three possibilities arise among the T_i, then obviously it will be necessary to eliminate at least one of the species a, b, c from X in order to obtain an agreement subtree. In this situation we will refer to $\{a, b, c\}$ as a *conflicted triple* of species.

An argument due to Bryant [**Bry98**] shows that our problem can be reduced in this way to the well-known 3-HITTING SET problem (see [**GJ79**]), that takes as input a collection $\mathcal{C}$ of 3-element subsets of a set X and a positive integer k, and must answer whether there is a subset $X' \subseteq X$ with $|X'| \leq k$ such that for each $A \in \mathcal{C}$, $A \cap X' \neq \emptyset$. Bryant's argument shows that our problem is equivalent to finding a k-element hitting set for the triples of X that are conflicted with respect to the T_i.

The set of conflicted triples could be computed by exhaustively computing the restrictions of the T_i for each 3-element subset of X, but this is not very efficient. In time $O(n \log n)$ it is possible to either determine that two trees are isomorphic, or identify a conflicted triple. Once a conflicted triple is identified, we can branch in a search tree based on 3 possibilities for resolving the conflict. For example, if the conflicted triple is $\{a, b, c\}$ then we create three branches in the search tree by deleting one element (say a) from X and from all of the trees T_i. We now recursively attempt to determine if the modified instance can be solved with $k' = k - 1$.

There will be at most $O(3^k)$ nodes in the search tree, and the running time due to each node is $O(rn \log n)$, which yields the claimed running time. □

The algorithm sketched above uses 3-Hitting Set implicitly as a means of solving the parameterized MAST problem. An improved but more elaborate *FPT* algorithm for this problem is described in [**BFRS98**], where we not only reduce MAST to 3-Hitting Set, but in turn we reduce 3-Hitting Set to Vertex Cover, to obtain an algorithm with $c = (1+\sqrt{17})/2$. Some further examples of the systematic adaptation of *FPT* algorithms into useful heuristics for problems in computational biology (where there is an abundance of naturally parameterized problems) can be found in [**FKS98**].

4.3. The Steiner Problem for Generalized Hypercubes. In this section we give another *FPT* result based on the method of reduction to a problem kernel. This leads to an important heuristic algorithm that is *already in use*. We consider a slightly more general problem than the one described in §1.

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 $i = 1, ..., n$, each described by a set of vertices V_i and a function
$$t_i : V_i \times V_i \to I\!N$$
(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 ith character.)
2. A positive integer k_1 such that $|V_i| \leq k_1$ for $i = 1, ..., n$.
3. A set X of k_2 length n vectors x_j for $j = 1, ..., k_2$, where the ith component $x_j[i] \in V_i$. That is, for $j = 1, ..., k_2$,
$$x_j \in \Omega = \prod_{i=1}^{n} V_i$$
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 1:1 with 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 4.6. THE STEINER PROBLEM FOR GENERALIZED HYPERCUBES *is fixed-parameter tractable.*

PROOF. We define an equivalence relation $i \sim j$ on the index space $\{1, ..., n\}$ 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 $v_1, ..., v_m$ denote the vertices of K_m. Let T be a rooted tree with k_2 leaves labeled from $v_1, ..., v_m$. 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 $\{v_1, ..., v_m\}$, 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 $l \sim_T l'$ for all such trees T.

For $i, i' \in \{1, ..., n\}$ 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. □

The parameter function for this simple kernelization algorithm is not very good and can probably be much improved. 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.

In §1 we reported 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 [**Fel97**]:

"That's what I already do!"

4.4. Kernelization and Heuristics: A Universal Connection. The example problems we have considered above exhibit a profound connection between kernelization algorithms for an *FPT* problem, and heuristics for the general unparameterized problem. The connection goes in both directions:

- An improved algorithm for the general problem, applied to the problem kernel, may extend the useful range of parameter values for which the problem is well solved.
- Because the kernelization phase simplifies and decreases the size of the problem instance, it is a reasonable first step for any general attack on the unparameterized problem.

It is interesting to investigate the question of to what extent good kernelization algorithms are typical of problems in *FPT*. This can be formalized as follows.

Definition 4.7. A parameterized problem L is *kernelizable* if there is there is a parametric transformation of L to itself that satisfies:

1. the running time of the transformation of (x, k) into (x', k'), where $|x| = n$, is bounded 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 4.8. *A parameterized problem L is in* FPT *if and only if it is kernelizable.*

Proof. The "if" direction is trivial. We can derive the "only if" direction from the result proved in [**CCDF97**] that a parameterized language L is in *FPT* if and only if there is a constant $\alpha \geq 1$ and a function $f(k)$ such that we can determine if $(x, k) \in L$ in time $O(n^\alpha + f(k))$. Our kernelization algorithm for L considers two cases:

1. If $k < f^{-1}(n)$ then in time $O(n^\alpha)$ we can simply answer the question.
2. If $k \geq f^{-1}(n)$ then $n \leq f(k)$ and the instance (x, k) already belongs to the problem kernel.

□

Since we are naturally interested in the *efficiency* with which problem kernels can be computed, it makes sense to consider the following modification of the above definition. This essentially sets up a time hierarchy inside of *FPT* based on the running time of the kernelization algorithm that every *FPT* problem has (by the lemma above).

DEFINITION 4.9. Let c be a fixed constant. A parameterized problem L is *strong n^c-kernelizable* if there is a recursive function f, a polynomial q and a parametric transformation of L to itself running in time $O(q(k)n^c)$ for $n = |x|$, taking (x, k) to (x', k') such that:

1. $k' \leq k$, and
2. $|x'| \leq f(k)$.

We say L is *weak n^c-kernelizable* if f is not required to be recursive. Let $K(n^c)$ denote the c-kernelizable subset of *FPT* (either strong or weak depending on the context).

The following somewhat technical theorem corresponds to the standard time hierarchy theorem, but is more difficult to prove because of the need for a "wait and see" argument to deal with the parameter functions.

THEOREM 4.10. *If $c < c'$ are constants, then*

$$K(n^c) \subset K(n^{c'}) \subset FPT$$

PROOF. It is sufficient to prove that $K(n^c) \subset FPT(n^c)$ for all $c \geq 1$, where by strong $FPT(n^c)$ we mean the parameterized problems solvable in time $f(k)n^c$ for some recursive function f. We will only do the case $c = 1$, the more general case follows by the same technique. We consider the strong flavor of the theorem first.

Let φ_e denote the e-th partial recursive function. Let g_e denote the e-th parametric self reduction, so that $g_e : (x, k) \mapsto (h_e(x), q_e(k))$, and is polynomial in k, say in time $|k|^e$, but linear in $|x|$. It suffices to construct a $FPT(n)$ language L to meet the requirements $\mathcal{R}_{e,i}$ below.

$$\begin{array}{ll} \mathcal{R}_{e,i}: & \text{either } \varphi_i \text{ is not total or} \\ & \exists x, k\ |h_e(x)| \not\leq \varphi_i(k) \text{ or} \\ & \exists x, k\ (g_e((x, k)) \in L \text{ iff } (x, k) \notin L). \end{array}$$

We set aside row $\langle e, i\rangle$ for the sake of requirement $\mathcal{R}_{e,i}$. Our language L will be in *FPT* with each row in time $2^k|x|$. At each stage s we will look only at one requirement, cycling through the requirements so that we see them infinitely often. At stage s we will decide the fate of all strings (x, k) with $|x| = s$. For a single requirement $\mathcal{R}_{e,i}$, our strategy is as follows. We need to do nothing until we arrive at a stage s where $\varphi_i(k) \downarrow$ in fewer than s steps. At this stage $\mathcal{R}_{e,i}$ becomes *active*. Note that if $\mathcal{R}_{e,i}$ never becomes active then φ_i is not total. (Similarly we can now check that $|h_e(x)| \leq \varphi_i(k)$, which will be implicit in the discussion below.)

We next consider an active requirement at a stage t where we can know that $2^{k-1}t > |k|^e t + \varphi_i(k)$. Note that at such a stage, we can, in $2^{k-1}t$ steps, work out the image of $(1^t, k)$ under g_e, namely $(h_e(1^t), q_e(k)) = (x', k')$, say. The diagonalization is then simple. If we see that we have already put (x', k') into L then we declare that (x, k) will be kept out of L. If we see that we have $(x', k') \notin L$ then we put (x, k) into L.

This concludes the strategy for a single requirement $\mathcal{R}_{e,i}$ and there is obviously no problem in dealing with the combinations of requirements.

In the arbitary case, where f is not computable, then we will replace the requirement $\mathcal{R}_{e,i}$ above by the infinitely many requirements $\mathcal{R}_{e,i,j}$ below.

$$\begin{aligned}\mathcal{R}_{e,i,j}: \quad & \text{either } \exists x, k h_e(x) \not\leq j \text{ or} \\ & \exists x, k\ (g_e((x,k)) \in L \text{ iff } (x,k) \notin L).\end{aligned}$$

The idea is that j is the "guess" we will use for $f(k)$. While we think that $f(k) = j$ we regard j as active. We activate $j+1$ when j proves wrong. If all the j prove wrong then there is no value for $f(k)$. We devote a single row $\langle e, i \rangle$ to the collection $\{\mathcal{R}_{e,i,j} : j \in \mathbb{N}\}$. The idea is essentially the same. When we consider j, we will be looking for stages t where $2^{k-1}t > |k|^e t + j$. If we see that the size of $h_e(1^t)$ exceeds j then we cancel j and move on to $j+1$. If not, then we can diagonalize as before. □

Cheeseman et al. [**CKT91**] have discussed (essentially) the idea that *reduction to a problem kernel* is a strategy that can and should be applied as a preprocessing step for *any* computational problem. They describe reduction rules for Graph Coloring, and present some statistical evidence that the set of reduced instances represent a complexity "phase transition" where the hard instances of the problem are concentrated.

4.5. ***FPT*** **Methods and Other Means of Coping.** There seem to be fairly strong methodological connections between *FPT* algorithm design methods and methods for devising polynomial-time approximation schemes and other approximation heuristics. Such a connection might naturally be expected, since in an approximation scheme for an *NP*-hard problem, one would normally anticipate a running time exponential in $k = 1/\epsilon$ for an algorithm that computes an approximation to within a factor of $(1 + \epsilon)$ of optimal. (This is isn't strictly necessary, since the problem may have a *fully polynomial-time approximation scheme* (see [**GJ79**]). However, these seem to be quite rare.) We mention three examples of connections between *FPT* methods and other coping strategies, especially approximation.

Example 4.1 (Efficient Approximation Schemes for Planar Graphs). A notable example of efficient approximation is the family of schemes devised by Baker [**Ba94**] for planar graph problems. The basic idea is quite simple and very much based on the standard *FPT* techniques of bounded treewidth and pathwidth (thoroughly exposited in [**DF98**]). For concreteness, consider the problem of computing a minimum dominating set for a

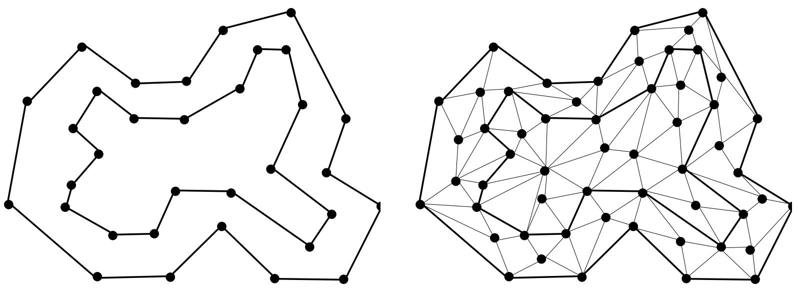

FIGURE 5. Illustration of bounded width onion decomposition for a planar graph. The figure on the left shows the boundaries of the layers.

planar graph. Starting from a planar embedding of the graph G, we can define a kind of "onion" decomposition of G. The layers of this decompostion can be defined in such a way that each has bounded treewidth. Figure 7 illustrates the idea.

The basic idea is to solve the problem exactly for each layer, using bounded treewidth machinery, and then combine the solutions for the layers. There is a small amount of inefficient overlap as the solutions for the layers are combined (by taking the union of the relevant vertex sets, for example). As the width of the layers is increased (at the usual *FPT* costs for bounded treewidth algorithms), the overall effect of the inefficient overlap on the goodness of the approximation progressively decreases, yielding an approximation to within a factor of $(1+\epsilon)$ of optimal for G for an appropriately chosen width of the layers. This same idea can be applied to a number of other graph problems.

EXAMPLE 4.2 (Testing Hereditary Properties Efficiently on Average). One of the few general techniques for devising algorithms that are efficient on average has been described by Gustedt and Steger [**GS94**] based on obstruction sets, one of the most powerful ideas in the *FPT* toolkit. The basic idea of their approach is that for some properties of combinatorial objects that are lower ideals in an appropriate partial order (which implies that the property is characterized by an obstruction set), the probability may be high for "typical" inputs that the answer is "no" (the object does not have the property), and that this sometimes can quickly be detected by checking for obstructions (not necessarily all of them). This strategy is employed as a preface, quickly answering the question for *most* inputs, thus raising the performance on average of whatever algorithm is used to deal with the remaining cases. Arguing along similar lines to our discussion of kernelization, they point out that this is a reasonable preface for *any* algorithmic attack on these kinds of problems. Our Algorithm 4.5 can be viewed in part as an example of this strategy. Langston et al. have similarly used approximate obstruction sets to design heuristics for VLSI layout problems [**LR91, GLR92**].

EXAMPLE 4.3 (Iterative Approximation Heuristics). An important example of a much-used approximation heuristic is the k-Opt heuristic for the TRAVELING SALESMAN PROBLEM [**JMc97**]. This can be viewed as a simple example of the following general recipe for the design of local search heuristics based on *FPT* methods.

First: Identify an *FPT* result for a modification of the problem that can be used as a subroutine to compute a single iteration of the local search. (For the k-Opt heuristic for the TSP, the relevant subroutine computes the best rearrangement of the k pieces into which the current best tour has been cut. This can be accomplished in time $O(k!n)$.)

Second: Iterate the application of this subroutine.

This design paradigm naturally raises the following question about the k-Opt neighborhood structure for TSP: Is there an *FPT* algorithm to compute the best neighbor s' of a solution s? Fisher [**Fis95**] has shown that it is *NP*-complete to determine whether there is a local optimum within k steps of s for the 2-Opt neighborhood structure for TSP. Whether this problem is in *FPT* is also open.

4.6. Some Further Discussion.

REMARK 4.11. The design of non-polynomial algorithms has been an area somewhat neglected by theoretical computer scientists — or at least it is an area in which there are many unexplored opportunities of importance to the applied community. The design of *FPT* algorithms systematically focuses attention in this direction. It is notable how the original, unparameterized problem is resurrected in the problem kernel (e.g., for VERTEX COVER in the second phase of the *FPT* algorithm we are essentially addressing the problem for a graph of size n with $k = \sqrt{n}$). It would be interesting to know if any systematic connection can be made between the "optimum" size of a problem kernel (which requires a definition) and the *complexity threshold* for the problem [**KS94**], or with the structural notion of the *complexity core* of the decision problem.

REMARK 4.12. For *FPT* problems that can be kernelized efficiently we end up in the happy situation that the problem can be solved for input of *any* size, so long as the parameter function $f(k)$ is not too large. This means that our reward for work on an improved algorithm for the general hard problem (with k unrestricted) is *leveraged* in a very significant way for problems that are *FPT* (by applying the improved general algorithm to the kernel). This also suggests measuring our success on an *FPT* problem L in terms of *c-klams*, where the c-klam value of an algorithm A is defined to be the largest $k_{\max}$ such that:

1. L can be solved by A in time $f(k) + n^c$, and
2. $f(k_{\max}) \leq U$, where U is some reasonable absolute bound on the maximum number of steps of any computation, e.g., $U = 10^{12}$ as

suggested by Yessenin-Volpin [**YV70**] (or perhaps a somewhat larger number).

If we can achieve for TREEWIDTH a $c = 3$ klam value of even just 8 or 10, this will be very significant for practical applications.

REMARK 4.13. The program of dealing with a hard problem by parameterizing can be compared to the program in Chapter 4 of Garey and Johnson [**GJ79**]of mapping the boundary between *NP*-hardness and polynomial-time subproblems in the following way. Suppose that we have an *FPT* result requiring time $O(2^k n^c)$ about the problem Π with parameter k. There are two different natural "forgetful functors" from the parametric to the classical framework. The first regards this *FPT* result as saying that for each fixed k the classical problem is solvable in polynomial time. The information loss in this is obvious, since it forgets entirely the distinction between *FPT* and *XP* (see Table 1). A second possible mapping of the *FPT* result to the classical framework would regard this as saying that Π restricted to $k \leq \log n$ is solvable in *P*-time. In general, it would be fruitful to find, for a collection of parameters of a problem Π, some such bounds (as functions of the input size n) such that for parameters less than these bounds, Π can be solved in polynomial time. (I.e., this is a nice canonical kind of subproblem.) But this is precisely the issue in parameterized complexity: for a given collection of parameters, is there *any* corresponding collection of bounds (as functions of n) such that the canonical subproblem defined by these bounds is in *P*? $W[1]$-hardness shows when the answer is probably "no". Improving the parameter function of an *FPT* problem is equivalently the problem of improving such bounds.

REMARK 4.14. We conjecture that the most promising method that the reader can apply towards developing useful *FPT* algorithms and heuristics for *NP*-hard problems is to purchase the moderately priced and entertaining book [**DF98**].

5. Applications of Treewidth to Problems of Logic

In this section we explore the theme of treewidth as a hidden parameter in many problems, particularly those of logic and programming languages. It is possible that treewidth represents one of the most universal hidden economies of computational tractability. (This section is unavoidably somewhat technical; the reader needs to be familiar with the "Myhill-Nerode" bounded treewidth algorithmic machinery developed in [**FL89, Co90, CL95, AF93**].)

Thorup recently proved the remarkable result that the flow graphs that arise from structured programs have bounded treewidth. The bound on the treewidth is specific to the particular programming language, depending on whether certain language constructions (such as loop-exits, conditional 'or's, etc.) are allowed. For example, the flow graphs of structured C++ programs have a treewidth bound of 7 [**Th97**].

We explore how treewidth can be applied as a parameter to two problems in logic, Minimum Axiom Set and STRIPS Planning. These problems are defined classically as follows.

Minimum Axiom Set

Instance: A finite set S of *sentences*, an *implication relation* R consisting of pairs (A, t) where $A \subseteq S$ and $t \in S$, and a positive integer k.

Question: Is there a set $S_0 \subseteq S$ with $|S_0| \leq k$ and a positive integer n such that if we define S_i for $1 \leq i \leq n$ to consist of exactly those $t \in S$ for which either $t \in S_{i-1}$ or there exists a set $U \subseteq S_{i-1}$ such that $(U, t) \in R$, then $S_n = S$?

Minimum Axiom Set with parameter k has been shown to be complete for $W[P]$ in [**DFKHW94**].

Propositional STRIPS Planning

Instance: A dimension n representing a number of atomic formulas called *conditions*, an *initial state vector* $S \in \{0,1\}^n$ indicating which of conditions are initially *true*, a goal vector $G \in \{0,1,*\}^n$ expressing the goal in terms of conditions that should be *true* (1), *false* (0) or *don't care* (*), and a collection $\mathcal{O}$ of *operators*, where each operator $o \in \mathcal{O}$ is a pair $o = (P, Q)$ with $P \in \{0,1,*\}^n$ expressing the *preconditions* of the operator in the natural way, and $Q \in \{0,1,*\}^n$ expressing the *postconditions* of the action. An operator $o = (P, Q)$ can be applied to the current state vector X if for all i, $1 \leq i \leq n$, $P[i] = a \in \{0,1\}$ implies $X[i] = a$, that is, the preconditions naturally expressed by P are met by the state vector X. The state vector X' resulting from the operation o in this situation is defined by:

1. $X'[i] = X[i]$ if $Q[i] = *$, and
2. $X'[i] = Q[i]$ otherwise.

Question: Is there a sequence of operations that can be applied, starting from the initial state vector S, and resulting in a state vector T such that

$$\forall i,\ 1 \leq i \leq n :\ G[i] = a \in \{0,1\} \rightarrow T[i] = a.$$

The Propositional STRIPS Planning problem was first described by Fikes and Nilsson [**FN71**] and was proved to be *PSPACE*-complete by Bylander [**Byl94**].

We must first define how the notion of treewidth can be applied to these problems. To do this we define two digraph pebbling problems to which these logic problems can be conveniently translated.

Digraph Pebbling

Instance: A digraph $D = (V, A)$ for which the vertex set V is partitioned $V = R \cup B$ into two classes, the red vertices of R and the blue vertices of B, and a positive integer k.

Question: Is it possible to pebble each vertex of the digraph according to the following set of rules?

1. Start with k pebbles placed on red vertices.

2. A blue vertex b can be pebbled if there is a pebbled vertex u such that $ub \in A$.
3. A red vertex r can be pebbled if for every vertex u such that $ur \in A$, u is pebbled.

If every vertex is red, then this is the same problem as DIRECTED FEEDBACK VERTEX SET and therefore DIGRAPH PEBBLING is *NP*-complete [**Karp72**]. The parameterized complexity of DIRECTED FEEDBACK VERTEX SET (with parameter k) is semi-famously open.

SIGNED DIGRAPH PEBBLING
Instance: A red/blue bipartite digraph $D = (V, A)$ for which the vertex set V is partitioned $V = R \cup B$ into two classes, and also the arc set A is partitioned into two classes $A = A^+ \cup A^-$.
Question: Starting from the *start state* where there are no pebbles on any of the red vertices, is it possible to reach the *finish state* where there are pebbles on all of the red vertices, by a series of moves of the following form?
A legal move: If b is a blue vertex for which $\forall u$ such that $ub \in A^+$, u is pebbled, and $\forall u$ such that $ub \in A^-$, u is not pebbled (in which case we say that b is *enabled*), then the set of vertices v such that $bv \in A+$ are reset by making them all pebbled, and the set of vertices v such that $bv \in A^-$ are reset by making them all unpebbled.

We leave it to the reader to verify that an instance $\mathcal{I}$ of MINIMUM AXIOM SET can be naturally translated into an instance $\mathcal{I}'$ of DIGRAPH PEBBLING, where the set of sentences S of $\mathcal{I}$ becomes the set of blue vertices B of $\mathcal{I}'$. We can define the *treewidth* of $\mathcal{I}$ to be the treewidth of the digraph of $\mathcal{I}'$. The representation of a PROPOSITIONAL STRIPS PLANNING instance as an instance of SIGNED DIGRAPH PEBBLING is equally straightforward, and we adopt a similar notion of the treewidth of an instance.

We now consider the following parameterizations of these problems. By DIGRAPH PEBBLING I we refer to this problem with the parameter k. (This is complete for $W[P]$ by the results on MINIMUM AXIOM SET cited above.) By DIGRAPH PEBBLING II we refer to this problem with parameter (k, w) where w is the treewidth of the instance. By SIGNED DIGRAPH PEBBLING I we refer to this problem with parameter w, where w is the treewidth of the instance, and by SIGNED DIGRAPH PEBBLING II we refer to this problem with parameter (k, w) where w is the treewidth, and where k is a bound on the number of moves in which to reach the finish state. Note that the number of moves is a reasonable parameter for planning problems. The new results that follow all make use of the Myhill-Nerode techniques for bounded treewidth developed in [**MP94, FL89, AF93, BFW92, FHW93, DF98**].

We remark that it appears difficult to express either DIGRAPH PEBBLING II or SIGNED DIGRAPH PEBBLING II in Monadic Second-Order logic, a very powerful but not always applicable method for obtaining tractability results for bounded treewidth (see [**Co90**] and [**DF98**] for expositions).

In order to apply the Myhill-Nerode machinery for bounded treewidth the key notion that is needed is that of a *t-boundaried graph*, which is just a graph with t distinguished vertices. What is ultimately going on is that the structure of a bounded treewidth graph or digraph is represented by a rooted labeled *parse tree*. A graph of treewidth bounded by t can be parsed in linear time (i.e., this problem is linear *FPT*) by a theorem of Bodlaender [**Bod96**]. If $\mathcal{F}$ is a set of rooted labeled trees, then leaf-to-root tree automata provide a notion of finite-state recognizability for $\mathcal{F}$. The Myhill-Nerode theorem for bounded treewidth relates finite-state recognizability of $\mathcal{F}$ to the amount of information that must flow across a t-sized boundary in order to determine membership in $\mathcal{F}$. The machinery is applicable not just to ordinary graphs, but also to finitely edge- and vertex-colored digraphs.

THEOREM 5.1. DIGRAPH PEBBLING II *is in FPT.*

PROOF. It is sufficient to argue that the canonical equivalence relation on t-boundaried (vertex-colored) digraphs, for the family $\mathcal{F}_k$ of k-pebblable digraphs, has a finite number of equivalence classes for each fixed pair (t, k). We first review the essential notions.

By a *t-boundaried digraph* we mean a digraph $D = (V, A)$ equipped with t distinguished vertices labeled $1, ..., t$. We refer to these vertices as constituting the *boundary set* $\partial(D) \subseteq V$. The *interior* vertices of D are those vertices that are not boundary vertices, and we write $\text{int}(D) = V - \partial(D)$.

If D and D' are t-boundaried digraphs, we say that they are *compatible* if their boundaries are colored in the same way, that is, for $i = 1, ..., t$ the ith boundary vertex of D is red (blue) if and only if the ith boundary vertex of D' is red (blue).

If D and D' are compatible t-boundaried digraphs, then by $D \oplus D'$ we denote the digraph obtained by identifying the vertices $i \in \partial(D)$ with $i \in \partial(D')$ for $i = 1, ..., t$. The canonical equivalence relation on the set of all 2-vertex-colored t-boundaried digraphs, for a family of such digraphs $\mathcal{F}$, is defined for compatible digraphs by:

$$D \sim_{\mathcal{F}} D' \text{ if and only if } \forall D'' : \; D \oplus D'' \in \mathcal{F} \leftrightarrow D' \oplus D'' \in \mathcal{F}$$

Fix k and t. Our approach is to define a different (easier to work with) equivalence relation $\sim$ on t-boundaried digraphs, and then to show two things:
Claim 1. that $\sim$ has finite index, and
Claim 2. that $D \sim D'$ implies $D \sim_{\mathcal{F}} D'$, where $\mathcal{F} = \mathcal{F}_k$.
This allows us to conclude that $\sim_{\mathcal{F}_k}$ has finite index on t-boundaried digraphs.

Our definition of $\sim$ is based on a set of abstract *tests* in the sense of the "method of test sets" developed and exposited in [**FL89, AF93, CDDFL98, DF98**].

A *test* T is specified by the following information:

1. A partition of $\{1, ..., t\}$ into two subsets, the red subset T_R and the blue subset T_B.
2. A *boundary starter set* $S \subseteq \{1, ..., t\}$. (We will use k_0 to denote the size of S.)

3. A positive integer k_1 such that $k_0+k_1 \leq k$. (We will use k_2 to denote $k-k_0-k_1$.)
4. A permutation π of $\{1,...,t\}-S$.
5. A subset $Q \subseteq T_B$.

A test T is *compatible* with a t-boundaried red/blue digraph D if the partition given in (1) accords with the colors of the boundary vertices of D.

Let $\mathcal{T}$ denote the set of tests. We have the bound on the size of the test set

$$|\mathcal{T}| \leq k2^{2t}t!$$

For a digraph $D=(V,A)$ and a vertex $v \in V$, let

$$\text{pred}(v) = \{u \in V : uv \in A\}$$

be the set of *predecessors* of v.

By a *pebbling sequence* for a t-boundaried digraph D we mean a permutation σ of the vertices of D. If σ is a pebbling sequence for D and v is a vertex of D, then v is *legal* with respect to σ if either of the following statements holds:

1. v is a blue vertex, and at least one predecessor of v precedes v in σ.
2. v is a red vertex, and every predecessor of v precedes v in σ.

Say that a t-boundaried digraph $D=(V,A)$ *passes* the test $T \in \mathcal{T}$ if T is compatible with D and there is a pebbling sequence σ for D that satisfies the requirements:

1. The prefix σ' consisting of the first k_0+k_1 vertices of σ includes the k_0 vertices of S and k_1 additional interior vertices of D.
2. The order in which the vertices of $\partial(D)-S$ occur in σ is specified by the permutation π of $\partial(D)-S$.
3. Every red vertex not in the prefix σ' is legal.
4. Every blue vertex b not in the prefix σ' is either legal or belongs to Q.

Intuitively, the set Q represents blue vertices that will be pebbled for "reasons on the other side of the boundary".

Let $\mathcal{T}(D)$ denote the set of tests passed by D. Given two t-boundaried digraphs D and D', define $D \sim D'$ if and only if $\mathcal{T}(D)=\mathcal{T}(D')$. Since the relation $\sim$ is defined in terms of equality of sets, it is clearly an equivalence relation.

Now that we have defined a suitable equivalence relation, we have only to prove the two Claims listed above. Claim 1 is trivial, since $\mathcal{T}$ is finite.

Proof of Claim 2. We argue by contraposition. Suppose $D \not\sim_{\mathcal{F}} D'$, and let this be witnessed by a t-boundaried digraph X with $D \oplus X \in \mathcal{F}$ and $D' \oplus X \notin \mathcal{F}$. Let σ denote a permutation of the vertices of $D \oplus X$ corresponding to a successful k-pebbling. Thus every vertex u occuring in σ after the initial prefix of length k is legal with respect to $D \oplus X$. Let S denote the starter set of size at most k for the successful pebbling (consisting of both interior and boundary vertices), and let S_0 denote the subset of S on the boundary

$$S_0 = S \cap \partial(D \oplus X)$$

with $k_0 = |S_0|$. Let k_1 denote the size of $S \cap \mathrm{int}(D)$. Let π denote the permutation of $\partial(D) - S_0$ inherited as a subsequence of σ. Then D passes the test T defined by the information: (1) the color partition of the boundary, (2) the boundary starter set S_0, (3) the positive integer k_1, (4) the permutation π, and (5) the set Q consisting of those blue vertices of the boundary which are illegal for σ restricted to the vertices of D.

We argue that D' fails T. Suppose not, and let σ' be a permutation of the vertices of D' that starts with S_0 followed by k_1 interior vertices of D' that is a valid pebbling sequence for D' showing that it passes the test T. Using σ' we can modify σ to obtain a valid pebbling sequence for $D' \oplus X$ and reach a contradiction. The contradiction will allow us to conclude that D' fails T and thus $D' \not\sim D$, which will establish Claim 2.

The modification of σ is described as follows. The permutation σ can be partitioned into three subsequences σ_0, σ_1 and σ_2 according to membership in the three disjoint sets of vertices $\partial(D \oplus X)$, $\mathrm{int}(D)$ and $\mathrm{int}(X)$. The modification consists of two steps:

1. The subsequence σ_1 is deleted.
2. Substrings of σ' are inserted.

The relevant substrings of σ' are those consisting only of interior vertices of D' that are bounded by either the initial block of starter vertices for the pebbling of D' described by σ' or by vertices of the boundary, or by the end of the sequence. Each such substring of interior vertices of D' has a left boundary and a right boundary. Call the substring whose right boundary is the end of σ' the *last substring.* All of the other substrings have a right boundary consisting of a boundary vertex of D'.

These substrings, other than the last substring, should be inserted into the modified σ just before each boundary vertex in σ, and the last substring should be concatenated to the end of the modified result. The block of interior starter vertices of σ' should be concatenated to the beginning of the modified σ. Call the result of all this σ''.

The point of all these modifications is to produce a pebbling sequence σ'' that works for $D' \oplus X$. What we have done is replace "what happened inside of D" according to σ, with equivalent bits from σ' describing how D' can pass the test T, which simply records "what D saw of X" in the pebbling described by σ (where D interacts with X only across the boundary). It is not hard to check that σ'' is a valid pebbling sequence for $D' \oplus X$. □

Figure 6 illustrates the argument. The boundary vertices of the two digraphs D and D' are indicated in bold in Figure 6(i) and Figure 6(ii). The boundary size is $t = 2$. We will take $k = 1$ It happens to be a fact that $D \sim D'$. The sequence $\sigma = (a, b, c, 1, v, 2, d, u)$ is a valid pebbling sequence for $D \oplus X$. The test T that corresponds to this is specified by:

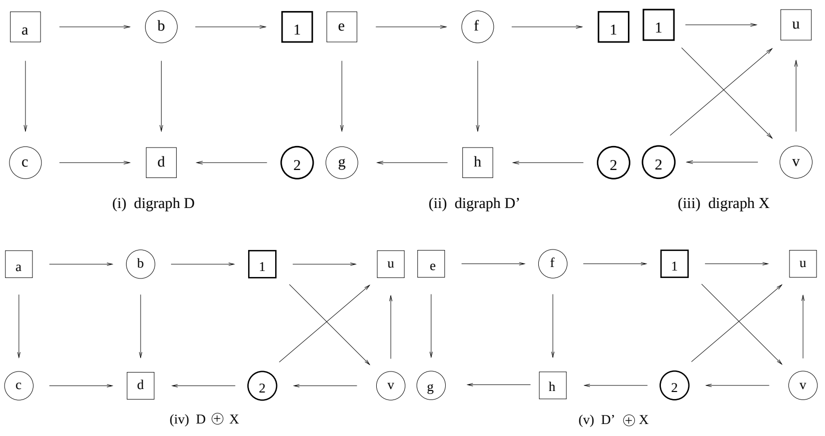

FIGURE 6. Examples for Theorem 5.1.

(1) the obvious partition of the boundary, (2) $S_0 = \emptyset$, (3) $k_1 = 1$, (4) $\pi = (1, 2)$, and (5) $Q = \{2\}$. Note that D' passes the test T with the sequence $\sigma' = (e, f, 1, 2, h, g)$. Using this to modify σ as described we get $\sigma'' = (e, f, 1, v, 2, u, h, g)$, a valid pebbling sequence for $D' \oplus X$.

REMARK 5.2. The algorithm implicitly described by Theorem 5.1 consists of two parts: (1) the computation of a tree decomposition, and (2) the evaluation of the resulting parse tree by a finite-state leaf-to-root tree automaton. The time complexity arising from (1) is bounded by the running time of Bodlaender's algorithm, $O(2^{35w^3}n)$ for a graph on n vertices and treewidth w. Given the tree decomposition, part (2) consists only of making finite-state transitions, which is fast. The problem is that the automaton that represents the algorithm may be very large. Based on the proof of the theorem, the best we can say is that the automaton has $O(2^{k2^{2^w}w!})$ states. This means that the practical difficulty is more likely to be about *space* than *time*, although this bound on the number of states is undoubtedly extremely pessimistic.

THEOREM 5.3. SIGNED DIGRAPH PEBBLING II *is in FPT.*

SKETCH. This is proved in much the same way, the key point being that there is a finite set of abstract tests that define an equivalence relation on the edge- and vertex-colored digraphs that refines the canonical equivalence relation. How to define these tests is the essential matter.

Since only k moves are to be made, the boundary would be in a sequence of at most k different states in the course of a solution. Note that there are at most 2^t different states that the boundary set can be in, in terms of which vertices of the boundary set are pebbled. A suitable test is specified by the sequence of boundary states (including information on which blue vertices of the boundary are "enabled" for a move by the states of their predecessors in the interior of D), and a budget of some number of moves (always less

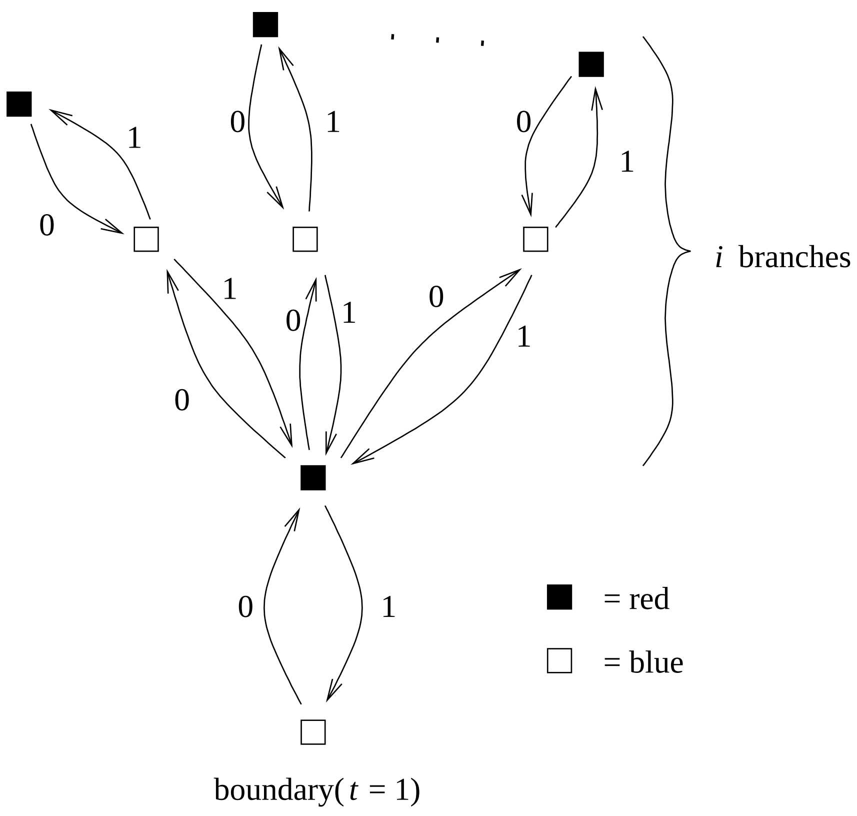

FIGURE 7. The trees for Theorem 5.4

than k) to get from one state to the next in the sequence, together with a specification of at most k changes of the boundary state due to actions on the other side. Call all this information a *plan* for a k-step pebbling of the digraph. The number of plans is bounded by a function of t and k. The question that defines the test for a digraph D is whether D can realize this plan, ending up with all of its red interior vertices pebbled.

The size of an appropriate set of tests $\mathcal{T}$ can be bounded by $|\mathcal{T}| \leq 3^{2kt}k^k$, which leads to a crude bound on the size of the resulting automaton of $O(2^{3^{2kt}k^k})$ states. □

THEOREM 5.4. SIGNED DIGRAPH PEBBLING *is not finite-state for any fixed treewidth bound $t \geq 1$.*

PROOF. Given the machinery, the argument is quite simple. Recall the usual argument using the Myhill-Nerode theorem that the formal language $L = \{a^n b^n : n \geq 0\}$ is not regular (i.e., finite-state recognizable by a linear automaton). The argument consists simply in exhibiting an infinite set of words x_i such that if $i \neq j$ then there is a word $y_{i,j}$ with $x_i y_{i,j} \in L$ and $x_j y_{i,j} \notin L$. One can take $x_i = a^i$ for $i = 1, 2, ...$ and $y_{i,j} = b^i$. We make here a completely analogous argument. (Other examples for graph problems can be found in [**AF93, BFW92, FHW93, DF98**].) We describe an infinite family of trees T_i for $i = 1, 2, \ldots$ such that if $i \neq j$, then there is a compatible tree $T_{i,j}$ such that:

1. $T_i \oplus T_{i,j}$ is a "yes" instance for the problem.
2. $T_j \oplus T_{i,j}$ is a "no" instance for the problem.

The trees T_i are shown in Figure 7. Each tree T_i consists a single blue boundary vertex connected to a red vertex to which i pendant paths of length 2 are attached. We can take $T_{i,j} = T_i$.

The reader can easily verify that 1 and 2 hold. □

We suspect that SIGNED DIGRAPH PEBBLING I is $W[1]$-hard. (It may even be that SIGNED DIGRAPH PEBBLING is NP-hard for trees, which would imply $W[P]$ hardness.) It seems that the *length of the plan* is an important parameter on which to focus on for the STRIPS PLANNING problem, in order to obtain tractable restrictions of the problem.

6. A Working Guide to Parametric Intractability

The main classes of parametric complexity are described in the tower:

$$FPT \subseteq W[1] \subseteq W[2] \subseteq \cdots \subseteq W[SAT] \subseteq W[P] \subseteq AW[P] \subseteq XP$$

The focus of this survey is on the analysis of concrete computational problems. For this purpose, all of the beautiful structural questions associated with the degrees of parameterized complexity, and even the definitions of most of the classes in the above hierarchy, are not very important. Suffice it to say that the $W[t]$ classes have natural complete problems that are variants of SATISFIABILITY. For concrete problem analysis, only the three classes

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

have compelling practical significance, where XP is the family of parameterized languages that can be recognized in time $O(f(k)n^{g(k)})$ where f and g are arbitrary functions. Knowing that a problem is in XP has some practical value and can be difficult to show. For example, for UNIT TASK LENGTH PRECEDENCE CONSTRAINED k-PROCESSOR SCHEDULING it is a famous open problem (discussed in [**GJ79**]) whether this is in XP or whether it is NP-complete for any fixed k, although it is proved in [**BF95**] be hard for $W[2]$.

Table 3 exhibits some of the many results that are now known about the complexity of naturally parameterized problems. Those problems which are only known to be hard for a parameterized class (as opposed to complete) are annotated as such. (The problem definitions can be found in Garey and Johnson [**GJ79**] and also [**DF98**] where there is a comprehensive inventory of known results.) The problems in the table are all known to belong to XP, with the exception of k-PROCESSOR SCHEDULING.

$W[P]$	Linear Inequalities	**[ADF95]**
	Minimum Axiom Set	**[DFKHW94]**
	Short Satisfiability	**[ADF95]**
	Weighted Circuit Satisfiability	**[ADF95]**
$W[SAT]$	Weighted Satisfiability	**[ADF95]**
$W[t]$, for all t	Longest Common Subsequence ($k =$ number of seqs.,$\|\Sigma\|$) (hard)	**[BDFHW95]**
	Feasible Register Assignment (hard)	**[BFH94]**
	Triangulating Colored Graphs (hard)	**[BFH94]**
	Bandwidth (hard)	**[BFH94]**
	Proper Interval Graph Completion (hard)	**[BFH94]**
	Weighted t–Normalized Satisfiability	**[DF95a]**
$W[2]$	Weighted $\{0, 1\}$ Integer Programming	**[DF95a]**
	Dominating Set	**[DF95a]**
	Tournament Dominating Set	**[DF95c]**
	Unit Length Precedence Constrained Scheduling (hard)	**[BF95]**
$W[1]$	Shortest Common Supersequence (k) (hard)	**[FHK95]**
	Maximum Likelihood Decoding (hard)	**[DFVW98]**
	Weight Distribution in Linear Codes (hard)	**[DFVW98]**
	Nearest Vector in Integer Lattices (hard)	**[DFVW98]**
	Short Permutation Group Factorization (hard)	**[CCDF96]**
	Short Post Correspondence	**[CCDF96]**
	Weighted q–CNF Satisfiability	**[DF95b]**
	Vapnik–Chervonenkis Dimension	**[DEF93]**
	Longest Common Subsequence (length m common subseq. for k seqs., parameter (k, m))	**[BDFW95]**
	Independent Set	**[DF95b]**
	Square Tiling	**[CCDF96]**
	Monotone Data Complexity for Relational Databases	**[DFT96]**
	k-Step Derivation for Context Sensitive Grammars	**[CCDF96]**
	Clique	**[DF95b]**
	Short NTM Computation	**[CCDF96]**
FPT	Feedback Vertex Set	**[DF95c]**
	Graph Genus	**[RS85]**
	Minor Order Test	**[RS85]**
	Treewidth	**[Bod96]**
	Vertex Cover	**[BFR98]**

Table 3. A Sample of Parametric Complexity Classifications

6.1. The Nature of the Evidence for Parametric Intractability. As in the theory of *NP*-completeness, there are two kinds of evidence indicating, when a parameterized problem is hard for $W[1]$, that it is unlikely to be fixed-parameter tractable. The first is that given a sufficient amount of unsuccessful effort to demonstrate tractability for various problems in a class, the knowledge that a problem is hard for the class offers a cautionary sociological message, of the sort depicted in the famous cartoon in the opening pages of [**GJ79**]. As shown by Table 3, the amount of evidence of this sort is now substantial.

A second reason for the belief that $W[1]$-hardness implies parametric intractability, is rooted in the following fundamental theorem [**DFKHW94, CCDF96**].

Theorem 6.1 (Downey-Fellows). *The* k-Step Halting Problem for Nondeterministic Turing Machines *is complete for* $W[1]$.

On input consisting of a Turing machine M and a positive integer k (with k being the parameter), the question is whether M can reach a halting configuration in at most k steps. This problem is so generic and opaque that it is hard to imagine that there is any algorithm for it that radically improves on simply exploring the n-branching depth k tree of allowed transitions exhaustively. The theorem can be viewed as essentially a miniaturization of Cook's Theorem.

In general, W-hardness results tend to be harder to prove than theorems about *NP*-completeness, because of the control requirement concerning the parameter. Clique is a typical starting point for $W[1]$-hardness arguments. It is interesting that most natural parameterized problems seem to belong to a small number of degrees (*FPT*, $W[1]$, $W[2]$, $W[P]$, $AW[*]$ and $AW[P]$; for details see [**DF98**]).

6.2. $W[1]$-Hard Means No Good PTAS. One might suspect that parameterized complexity is related to the complexity of approximation. A very good connection is supplied by the following theorem first proved by Bazgan [**Baz95**], and later independently by Cesati and Trevisan [**CT97**], strengthening an earlier result of Cai and Chen [**CC97**].

Definition 6.2. An approximation algorithm has an *efficient PTAS* if it computes a solution within a factor of $(1+\epsilon)$ of optimal in time $O(f(\epsilon)n^c)$ for some constant c.

Definition 6.3. For a maximization (resp. minimization) problem A, the induced language L_A is the parameterized language consisting of all pairs (x, k) where x is an instance of A and $\text{opt}(x) \geq k$ (resp. $\text{opt}(x) \leq k$).

Theorem 6.4 (Bazgan). *If A has an efficient PTAS then $L_A \in FPT$.*

Thus if the parameterized problem naturally associated with an optimization problem A is hard for $W[1]$, then A cannot have an efficient PTAS unless $FPT = W[1]$. It is worth noting that some (but by no means all)

NP-completeness reductions are serendipitously parametric and thus provide demonstrations of $W[1]$-hardness and non-approximability "for free". An important optimization problem that has a PTAS but is not known to have an efficient PTAS is the Euclidean Traveling Salesman Problem. The PTAS for this problem due to Arora runs in time $O(n^{30/\epsilon})$ [**Ar96**].

6.3. The Complexity of Propositional STRIPS Planning (for Short Plans) and k-Step Petri Net Reachability. We prove two negative complexity results about k-step Propositional STRIPS Planning and k-step Petri Net Reachability (for parameter k) for otherwise unrestricted inputs to these problems. These results should be interpreted as indicating that some sort of structure on the Petri nets and the set of planning options is a vital parameter for tractability of these problems.

Theorem 6.5. k-Step Propositional STRIPS Planning *is hard for* $W[1]$.

Proof. We reduce from the k-Step Halting Problem for Nondeterministic Turing Machines that is proved to be complete for $W[1]$ in [**CCDF96**]. Let the instance of the Halting Problem consist of the Turing machine $M = (Q, \Sigma, \delta, F)$ where Q is the set of states, Σ is the alphabet, δ is the transition relation, and F is the set of accept states, together with a positive integer k (the number of steps of a computation).

The dimension n of the instance of STRIPS Planning to which we reduce (M, k) is given by

$$n = |Q| + 1 + k + k|\Sigma|.$$

We can think of the components of the vectors of the STRIPS Planning as belonging to $k + 3$ blocks. The first block of size $|Q|$ will be constrained to have exactly one component that is a "1" with all the rest "0", indicating the state of M. The second block of size 1 indicates whether the state indicated by the "1" component of the first block is an accept state of M. The third block of size k indicates the position of the tape head of M. The k remaining blocks, each of size $|\Sigma|$, indicate the symbols on the k tape squares that might be visited during a k-step computation.

Each possible transition of M:

$$(q, a) \vdash (p, b, X) \text{ where } X \in \{L, R\}$$

becomes k different STRIPS operators (one for each of the locations on the tape where the transition may occur).

We leave the remaining details to the reader. $\square$

Theorem 6.6. k-Step Petri Net Reachability *is hard for* $W[1]$.

For the definition of the Petri Net Reachability problem we refer the reader to [**Pet81**].

PROOF. The reduction from k-STEP STRIPS PLANNING to k-STEP PETRI NET REACHABILITY is essentially trivial, using two places of the Petri net to represent each component of the planning vectors. □

Showing that these problems belong to $W[1]$ is straightforward and can be done in much the same manner as the proof that the k-STEP HALTING PROBLEM FOR NONDETERMINISTIC TURING MACHINES is in $W[1]$ (see [**CCDF96**]).

6.4. The Upper Reaches of Parametric Complexity. There are a number of naturally parameterized problems, such as GRAPH k-COLORING that are *NP*-complete for fixed values of k. Does the theory of parameterized complexity have anything to say about these? Since this is a well-defined parameterized problem, it has to be located *somewhere* in the parametric degree structure. The following theorem shows that the hierarchy of parameterized complexity classes seems to disappear into clouds of unknowing at the top end.

THEOREM 6.7. *The naturally parameterized* GRAPH k-COLORING *problem has an imponderable relationship to* XP *in the following sense:*

1. *If* GRAPH k-COLORING *is in XP then $P = NP$.*
2. *If* GRAPH k-COLORING *is hard for XP then $P \neq NP$.*

PROOF. Clearly, if GRAPH k-COLORING belongs to *XP* then $P = NP$. On the other hand, if it is hard for *XP*, then $P \neq NP$ by the following argument. For each k, let L_k be a problem that is complete for DTIME(n^k) by a polynomial-time reduction, such as the k-MICE CAT AND MOUSE GAME problem [**DF98**]. Consider the parameterized language

$$L = \{(y, k) : y \in L_k\}.$$

It is obvious that $L \in XP$. Suppose that there is a parametric reduction of L to GRAPH k-COLORING. Thus from (y, k) we can produce a graph G of size bounded by $f(k)n^c$ and an integer $k' = g(k)$ such that G is k' colorable if and only if $(y, k) \in L$, i.e., $y \in L_k$, where f, g are some fixed but arbitrary functions, c is a constant and $n = |y|$. The time required for the transformation can also be assumed to be bounded by $f(k)n^c$.

If $P = NP$ then there is an algorithm running in time $O(n^{c'})$ that can decide if G is k'-colorable, and thus we have a way of determining whether $y \in L_k$ in time bounded by $h(k)n^{c''}$ for some function h and constant c''. When k and n are sufficiently large this contradicts the DTIME hierarchy theorem. □

Thus we are unable to say anything about the relationship of GRAPH COLORING and *XP* without settling the question of whether $P = NP$.

7. The Role of Parameterized Complexity Analysis

The current approach to the analysis of concrete computational problems is dominated by two kinds of effort:

(1): The search for asymptotic worst-case polynomial-time algorithms.
(2): Alternatively, proofs of classical hardness results, particularly *NP*-hardness.

We expect that these will become substantially supplemented by:

(1′): The design of *FPT* algorithms for various parameterizations of a given problem, and the development of associated heuristics.
(2′): Alternatively, demonstrations of $W[1]$-hardness.

We think this will happen because we are inevitably forced towards something like an *ultrafinitist* [**YV70**] outlook concerning computational complexity because of the nature of the universe of interesting yet feasible computation. The main point of this outlook is that numbers in different ranges of magnitude should be treated in qualitatively different ways.

The pair of notions (1′) and (2′) are actually rather straightforward mutations of (1) and (2), and they inherit many of the properties that have made the framework provided by (1) and (2) so successful. We note the following in support of this position.

- The enrichment of the dialogue between practice and theory that parameterized complexity is based on always makes sense. It *always* makes sense to ask the users of algorithms, "Are there aspects of your problem that may typically belong to limited distributional ranges?"
- Fixed-parameter tractability is a more accurate notion of "the good". If you were concerned with inverting very large matrices and could identify a bounded structural parameter k for your application that allows this to be done in time $O(2^k n^2)$, then you might well prefer this *classically exponential-time* algorithm to the usual $O(n^3)$ polynomial-time algorithm.
- The "bad", $W[1]$-hardness, is based on a miniaturization of Cook's Theorem in a way that establishes a strong analogy between *NP* and $W[1]$. Proofs of $W[1]$-hardness are generally more challenging than *NP*-completeness, but it is obvious by now (see Table 3) that this is a very applicable complexity measurement.
- Problems that are hard do not just go away. Parameterization allows for several kinds of sustained dialogue with a single problem, in ways that allow finer distinctions about the causes of intractability (and opportunities for practical algorithms, including systematically designed heuristics) to be made than the exploration of the "*NP*-completeness boundary" described in [**GJ79**].
- Polynomial time has thrived because of the empirical circumstance that when polynomial-time algorithms can be devised, one almost always has small exponent polynomials. This is also true for *FPT* algorithms.
- Polynomial time is robust in that it seems to support a strong form of Church's thesis, i.e., that polynomial time on Turing machines is the

same as polynomial time on any reasonable computing device. This also seems to be true for *FPT*.

- Polynomial time has thrived because it is a mathematically rich and productive notion allowing for a wide variety of algorithm design techniques. *FPT* seems to offer an even richer field of play, in part because it encompasses polynomial time as usually the best kind of *FPT* result. Beyond this, the *FPT* objective encompasses a rich and distinctive positive toolkit, including novel ways of defining and exploiting parameters.
- There is good evidence that not only are small polynomial exponents generally available when problems are *FPT*, but also that simple exponential parameter functions such as 2^k are frequently achievable, and that many of the problems in *FPT* admit kernelization algorithms that provide useful start-ups for *any* algorithmic attack on the problem.
- The complexity of approximation is handled more elegantly than in the classical theory, with $W[1]$-hardness immediately implying that there is no efficient PTAS. Moreover, *FPT* algorithm design techniques appear to be fruitful in the design of approximation algorithms.
- Parameterization is a very broad idea. It is possible to formulate and explore notions such as randomized *FPT* [**FK93**], parameterized parallel complexity [**Ces96**], parameterized learning complexity [**DEF93**], parameterized approximation [**BFH97**], parameterized cryptosystems based on $O(n^k)$ security, etc.

We feel that the parametric complexity notions, with their implicit ultrafinitism, correspond better to the natural landscape of computational complexity, where we find ourselves overwhelmingly among hard problems, dependent on identifying and exploiting thin zones of computational viability. Many natural problem distributions are generated by processes that inhabit such zones themselves (e.g., computer code that is written in a structured manner so that it can be comprehensible to the programmer), and these distributions then inherit limited parameter ranges because of the computational parameters that implicitly govern the feasibility of the generative processes, though the relevant parameters may not be immediately obvious. [2]

8. Some Open Problems

Parameterized complexity has been primarily motivated by concrete computational problems. The structure theory of parameterized complexity is very rich, and also not very well developed. One of the main questions is: how many of the important structural theorems of classical complexity have parameterized analogs? We know that there is a good analog of Cook's Theorem [**CCDF96**], and there is a good analog of Mahaney's Theorem

[2]For a philosophically similar discussion see [**Gur89**].

on sparse hard sets [**CF96**]. There is a partial analog of Ladner's Density Theorem [**DF93, DF98**]. Notably lacking are parameterized analogs of probability amplification. An analog of Toda's Theorem [**To91**] would be very interesting, as together with the results of [**DFR98a**] it would show that UNIQUE k-CLIQUE is hard for $W[t]$ for all t via randomized parametric reductions.

***FPT* or W-Hard?** The following concrete problems seem important to classify because of their significant applications.

1. The naturally parameterized DIRECTED FEEDBACK VERTEX SET problem. This can be shown to be equivalent to the naturally parameterized DIRECTED FEEDBACK ARC SET problem. For undirected graphs the problem is in *FPT*.
2. The problem of determining whether it is possible to delete at most k edges of a graph so that the resulting graph is bipartite.
3. The problem of determining whether it is possible to delete at most k vertices from a graph so that the result is an interval graph. This has important applications in DNA sequence analysis [**Sha97**].
4. The problem of determining whether a graph H is immersed in a graph G, where the parameter k is the size of H. This is in *XP* and the immersion partial ordering of graphs is a well-partial-order by the major theorems of Robertson and Seymour [**RS85, RS96**]. GRAPH TOPOLOGICAL CONTAINMENT is a similarly interesting problem, currently known only to belong to *XP*.
5. The LONGEST COMMON SUBSEQUENCE and SHORTEST COMMON SUPERSEQUENCE problems for k sequences over an alphabet of constant size (as is the case in analyzing biological sequences). If the size is not constant and the parameter is the pair (s, k) where s is the size of alphabet and k is the number of sequences, then both problems are hard for $W[t]$ for all t [**BDFHW95, Hal96**].
6. The GRAPH ISOMORPHISM problem where the parameter k is the maximum degree of the graphs. This is in *XP* by the results of Luks [**Luks82**]. This problem would seem to be a reasonable candidate for representing a parameterized degree intermediate between *FPT* and $W[1]$, for the same reasons that GRAPH ISOMORPHISM classically seems to represent a polynomial time degree intermediate between *P* and *NP*. On the other hand, parameterized complexity behaves differently. TOURNAMENT DOMINATING SET, for example, which is classically intermediate, is precisely $W[2]$-complete [**DF95c**]. Maximum degree parameterized GRAPH ISOMORPHISM could even be in *FPT* without this entailing any known surprises. On the other hand, if it could be shown to be hard for $W[1]$, then this would give good evidence that GRAPH ISOMORPHISM in general is not in P.

It is also reasonable to wonder whether the parameterized complexity framework might provide new opportunities for addressing such central mysteries as the $P = NP$ question. For example, Theorem 6.7 (perhaps only weakly) suggests looking for a parametric reduction using an oracle for *NP* from the CAT AND MOUSE GAME to GRAPH COLORING. In general, it seems fruitful to explore the possibilities for stronger connections between the parameterized and classical frameworks.

Acknowledgments. We would like to thank Mike Hallett and Todd Wareham for stimulating discussions and assistance in improving the presentation. On a number of key points, the paper greatly benefited from the detailed comments and criticisms of an earlier draft by Bernard Moret and an anonymous referee, and from conversations with Chantal Korostensky of ETH.

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.

[AF93] K. Abrahamson and M. Fellows, "Finite Automata, Bounded Treewidth and Wellquasiordering," In: *Graph Structure Theory*, American Mathematical Society, Contemporary Mathematics Series, vol. 147 (1993), 539–564.

[AFGPR96] E. Allender, J. Feigenbaum, J. Goldsmith, T. Pitassi and S. Rudich, "The Future of Computational Complexity Theory: Part II," *SIGACT News* 27 (1996), 3–7.

[AK94] A. Amir and D. Keselman, "Maximum Agreement Subtree in a Set of Evolutionary Trees – Metrics and Efficient Algorithms," In: *35th Annual Symposium on Foundations of Computer Science*, 758–769, 1994.

[AL97] E. Aarts and J.K. Lenstra (eds.), *Local Search in Combinatorial Optimization*, John Wiley and Sons, 1997.

[ALMSS92] S. Arora, C. Lund, R. Motwani, M. Sudan and M. Szegedy, "Proof Verification and Intractability of Approximation Algorithms," *Proceedings of the IEEE Symposium on the Foundations of Computer Science*, 13–22, 1992.

[AMOV91] G. B. Agnew, R. C. Mullin, I. M. Onyszchuk and S. A. Vanstone, "An Implementation for a Fast Public-Key Cryptosystem," *J. Cryptology* 3 (1991), 63–79.

[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.

[AYZ94] N. Alon, R. Yuster and U. Zwick, "Color-Coding: A New Method for Finding Simple Paths, Cycles and Other Small Subgraphs Within Large Graphs," *Proc. Symp. Theory of Computing (STOC)*, ACM (1994), 326–335.

[Ba94] B. Baker, "Approximation Algorithms for *NP*-Complete Problems on Planar Graphs," *J.A.C.M.* 41 (1994), 153–180.

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

[BDFHW95] H. Bodlaender, R. Downey, M. Fellows, M. Hallett and H. T. Wareham, "Parameterized Complexity Analysis in Computational Biology," *Computer Applications in the Biosciences* 11 (1995), 49–57.

[BDFW95] H. Bodlaender, R. Downey, M. Fellows and H. T. Wareham, "The Parameterized Complexity of the Longest Common Subsequence Problem," *Theoretical Computer Science A* 147 (1995), 31–54.

[BF95] H. Bodlaender and M. Fellows, "On the Complexity of k-Processor Scheduling," *Operations Research Letters* 18 (1995), 93–98.

[BFH94] H. Bodlaender, M. R. Fellows and M. T. Hallett, "Beyond *NP*-completeness for Problems of Bounded Width: Hardness for the *W* Hierarchy," *Proc. ACM Symp. on Theory of Computing (STOC)* (1994), 449–458.

[BFH97] H. Bodlaender, M. Fellows, M. Hallett, "Parameterized Complexity and Parameterized Approximation for Some Problems About Trees," manuscript, 1997.

[BFR98] R. Balasubramanian, M. Fellows and V. Raman, "An Improved Fixed-Parameter Algorithm for Vertex Cover," *Information Processing Letters* 65 (1998), 163–168.

[BFRS98] D. Bryant, M. Fellows, V. Raman and U. Stege, "On the Parameterized Complexity of MAST and 3-Hitting Sets," manuscript, 1998.

[BFW92] H. Bodlaender, M. Fellows and T. Warnow, "Two Strikes Against Perfect Phylogeny," in: *Proceedings of the 19th International Colloquium on Automata, Languages and Programming*, Springer-Verlag, Lecture Notes in Computer Science vol. 623 (1992), 273–283.

[BGKRS95] R. S. Barr, B. L. Golden, J. P. Kelly, M. G. C. Resende and W. R. Stewart, "Designing and Reporting on Computational Experiments with Heuristic Methods," *J. Heuristics* 1 (1995), 9–32.

[BGS95] M. Bellare, O. Goldreich and M. Sudan, "Free Bit and Nonapproximability," *Proceedings of the 36th Annual IEEE Symposium on the Theory of Computing* (1995), 422-431.

[Bod96] H. Bodlaender, "A Linear Time Algorithm for Finding Tree Decompositions of Small Treewidth," *SIAM J. Comp.* 25 (1996), 1305–1317.

[Bry97] D. Bryant, "Building Trees, Hunting for Trees, and Comparing Trees," Ph.D. dissertation, Department of Mathematics, Univ. Canterbury, Christchurch, New Zealand, 1997.

[Bry98] D. Bryant, private communication, 1998.

[Byl94] T. Bylander, "The Computational Complexity of Propositional STRIPS Planning," *Artificial Intelligence* 69 (1994), 165–204.

[CC97] L. Cai and J. Chen. "On Fixed-Parameter Tractability and Approximability of *NP*-Hard Optimization Problems," *J. Computer and Systems Sciences* 54 (1997), 465–474.

[CCDF96] L. Cai, J. Chen, R. G. Downey and M. R. Fellows, "On the Parameterized Complexity of Short Computation and Factorization," *Arch. for Math. Logic* 36 (1997), 321–337.

[CCDF97] L. Cai, J. Chen, R. Downey and M. Fellows, "Advice Classes of Parameterized Tractability," *Annals of Pure and Applied Logic* 84 (1997), 119–138.

[CD94] K. Cattell and M. J. Dinneen, "A Characterization of Graphs with Vertex Cover up to Five," *Proceedings ORDAL'94*, Springer Verlag, Lecture Notes in Computer Science, vol. 831 (1994), 86–99.

[CDDFL98] K. Cattell, M. Dinneen, R. Downey and M. Fellows, "On Computing Graph Minor Obstruction Sets," *Theoretical Computer Science A*, to appear.

[CDM92] A. Colorni, M. Dorigo and V. Manniezzo, "An Investigation of Some Properties of an 'Ant Algorithm'," in R. Männer and B. Manderick (eds.) *Parallel Problem Solving From Nature 2*, North-Holland, Amsterdam (1992), 509–520.

[Ces96] M. Cesati, "Structural Aspects of Parameterized Complexity," Ph.D. dissertation, University of Rome, 1995.

[CF96] M. Cesati and M. Fellows, "Sparse Parameterized Problems," *Annals of Pure and Applied Logic* 62 (1996).

[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.

[Cl87] K. L. Clarkson, "New Applications of Random Sampling in Computational Geometry," *Discrete and Computational Geometry* 2 (1987), 195–222.

[CL95] B. Courcelle and J. Lagergren, "Equivalent Definitions of Recognizability for Sets of Graphs of Bounded Treewidth," *Mathematical Structures in Computer Science* (1996), 141–165.

[Co90] B. Courcelle, "Graph Rewriting: An Algebraic and Logical Approach," in: *Handbook of Theoretical Computer Science, vol. B*, J. van Leeuwen, ed., North Holland (1990), Chapter 5.

[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.*

[DEF93] R. Downey, P. Evans and M. Fellows, "Parameterized Learning Complexity," *Proc. 6th ACM Workshop on Computational Learning Theory* (1993), 51–57.

[Deu93] G. Deuck, "New Optimization Heuristics: the Great Deluge Algorithm and the Record-to-Record-Travel," *J. Computational Physics* 104 (1993), 86–92.

[DF93] R. Downey and M. Fellows, "Fixed Parameter Tractability and Completeness III: Some Structural Aspects of the W-Hierarchy," in: K. Ambos-Spies, S. Homer and U. Schöning, editors, *Complexity Theory: Current Research*, Cambridge Univ. Press (1993), 166–191.

[DF95a] R. G. Downey and M. R. Fellows, "Fixed Parameter Tractability and Completeness I: Basic Theory," *SIAM Journal of Computing* 24 (1995), 873–921.

[DF95b] R. G. Downey and M. R. Fellows, "Fixed Parameter Tractability and Completeness II: Completeness for W[1]," *Theoretical Computer Science A* 141 (1995), 109–131.

[DF95c] R. G. Downey and M. R. Fellows, "Parametrized Computational Feasibility," in: *Feasible Mathematics II, P. Clote and J. Remmel (eds.)* Birkhauser, Boston (1995) 219–244.

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

[DFKHW94] R. G. Downey, M. Fellows, B. Kapron, M. Hallett, and H. T. Wareham. "The Parameterized Complexity of Some Problems in Logic and Linguistics," *Proceedings Symposium on Logical Foundations of Computer Science (LFCS)*, Springer-Verlag, Lecture Notes in Computer Science vol. 813 (1994), 89–100.

[DFR98a] R. G. Downey, M. R. Fellows and K. W. Regan, "Parameterized Circuit Complexity and the W Hierarchy," *Theoretical Computer Science A* 191 (1998), 91–115.

[DFR98b] R. G. Downey, M. Fellows and K. Regan. "Threshold Dominating Sets and an Improved Characterization of $W[2]$," *Theoretical Computer Science A*, to appear.

[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.

[DFVW98] R. Downey, M. Fellows, A. Vardy and G. Whittle, "The Parameterized Complexity of Some Fundamental Problems in Coding Theory," *SIAM J. Computing*, to appear.

[DKL96] T. Dean, J. Kirman and S.-H. Lin, "Theory and Practice in Planning," Technical Report, Computer Science Department, Brown University, 1996.

[Due93] G. Dueck, "New Optimization Heuristics: the Great-Deluge Algorithm and the Record-to-Record Travel," *J. Computational Physics* 104 (1993), 86–92.

[Fel97] J. Felsenstein. Private communication, 1997.

[FHK95] M. Fellows, M. Hallett and D. Kirby, "The Parameterized Complexity of the Shortest Common Supersequence Problem," manuscript, 1995.

[FHW93] M. Fellows, M. Hallett and H. T. Wareham, "DNA Physical Mapping: Three Ways Difficult," in: *Algorithms – ESA'93: Proceedings of the First European Symposium on Algorithms*, Springer-Verlag, Lecture Notes in Computer Science vol. 726 (1993), 157–168.

[Fis95] S. T. Fischer, "A Note on the Complexity of Local Search Problems," *Information Processing Letters* 53 (1995), 69–75.

[FK93] M. Fellows and N. Koblitz, "Fixed-Parameter Complexity and Cryptography," *Proceedings of the 10th Intl. Symp. on Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, Springer-Verlag, Berlin, Lecture Notes in Computer Science vol. 673 (1993), 121–131.

[FKS98] M. Fellows, C. Korostensky and U. Stege, "Theory-Based Heuristics for Editing Gaps in Multiple Sequence Alignments," manuscript, 1998.

[FL87] M. Fellows and M. Langston, "Nonconstructive Proofs of Polynomial-Time Complexity," *Information Processing Letters* 26(1987/88), 157–162.

[FL88] M. Fellows and M. Langston, "Nonconstructive Tools for Proving Polynomial-Time Complexity," *Journal of the Association for Computing Machinery* 35 (1988) 727–739.

[FL89] M. R. Fellows and M. A. Langston, "An Analogue of the Myhill-Nerode Theorem and its Use in Computing Finite-Basis Characterizations," *Proceedings of the IEEE Symposium on the Foundations of Computer Science* (1989), 520–525.

[FL94] M. R. Fellows and M. A. Langston, "On Search, Decision and the Efficiency of Polynomial-Time Algorithms," *Journal of Computer and Systems Science* 49 (1994), 769–779.

[FN71] R. E. Fikes and N. J. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," *Artificial Intelligence* 2 (1971), 189–208.

[FPT95] M. Farach, T. Przytycka, and M. Thorup. "On the agreement of many trees" *Information Processing Letters 55* (1995), 297–301.

[Fr97] J. Franco, J. Goldsmith, J. Schlipf, E. Speckenmeyer and R.P. Swaminathan, "An Algorithm for the Class of Pure Implicational Formulas," to appear in *Discrete Applied Mathematics.*

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

[GLR92] R. Govindan, M. Langston and S. Ramachandramurthi, "A Practical Approach to Layout Optimization," *Proceedings of the International Conference on VLSI Design*, Bombay, India, January 1992.

[GM75] G. R. Grimmet and C. H. J. McDiarmid, "On Colouring Random Graphs," *Math. Proc. Cambridge Philos. Soc.* 77 (1975), 313–324.

[GM84] S. Goldwasser and S. Micali, "Probabilistic Encryption," *J. Computer and Systems Science* 28 (1984), 270–299.

[GS94] J. Gustedt and A. Steger, "Testing Hereditary Properties Efficiently on Average," *Orders, Algorithms and Applications: Proceedings of the International Workshop ORDAL'94*, Springer-Verlag, Lecture Notes in Computer Science, vol. 831 (1994), 100–116.

[Gur89] Y. Gurevich, "The Challenger-Solver Game: Variations on the Theme of *P=?NP*," *Bulletin EATCS* 39 (1989), 112–121.

[Hal96] M. Hallett. "An Integrated Complexity Analysis of Some Problems in Computational Biology," Ph.D. dissertation, Department of Computer Science, University of Victoria, 1996.

[Hart94] J. Hartmanis, "About the Nature of Computer Science," *Bulletin EATCS* 53 (1994), 170–190.

[Has96] J. Hastad, "Fast and Efficient Testing of the Long Code," *Proc. ACM Symposium on the Theory of Computing (STOC)* (1996).

[HGS98] M. Hallett, G. Gonnet, and U. Stege, "Vertex Cover Revisited: A Hybrid Algorithm of Theory and Heuristic," manuscript 1998.

[HL92] J. Hartmanis and H. Lin, editors, *Computing the Future: A Broader Agenda for Computer Science and Engineering*, National Academy Press, 1992.

[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.

[Hoch97] D. Hochbaum, "Various Notions of Approximations: Good, Better, Best and More," in: *Approximation Algorithms for NP-Hard Problems* (D. Hochbaum, ed.), PWS Publishing Co., Boston (1997), 346–398.

[Hoo95] J. N. Hooker, "Testing Heuristics: We Have It All Wrong," *J. Heuristics* 1 (1995), 33–42.

[IK75] O. H. Ibarra and C. E. Kim, "Fast Approximation for the Knapsack and Sum of Subset Problems," *J. Assoc. Computing Machinery* 22 (1975), 463–468.

[JDUGG74] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey and R. L. Graham, "Worst-Case Performance Bounds for Simple One-Dimensional Packing Algorithms," *SIAM J. Computing* 3 (1974), 299–325.

[Jo87] D. S. Johnson, "The *NP*-Completeness Column: An Ongoing Guide (19th edition)," *Journal of Algorithms* 8 (1987), 285–303.

[JM93] D. S. Johnson and C. McGeoch (eds.), *Network Flows and Matching: First DIMACS Implementation Challenge*, American Mathematical Society, 1993.

[JMc97] D.S. Johnson and L.A. McGeoch, "The Traveling Salesman Problem: A Case Study," in: Aarts and Lenstra (eds.), *Local Search in Combinatorial Optimization*, John Wiley and Sons, 1997.

[JPY88] D.S. Johnson, C.H. Papadimitriou and M. Yannakakis, "How Easy is Local Search?" *Journal of Computer and System Sciences* 37 (1988), 79–100.

[JT96] D. S. Johnson and M. Trick (eds.), *Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge*, American Mathematical Society, 1996.

[Karp72] R. M. Karp, "Reducibility Among Combinatorial Problems," in: *Complexity of Computer Applications*, Plenum Press, New York (1972), 85–103.

[Karp86] R. M. Karp, "Combinatorics, Complexity and Randomness," *Communications of the ACM* 29 (1986), 98–109.

[Kr90] M.W. Krentel, "On Finding and Verifying Locally Optimal Solutions," *SIAM Journal on Computing* 19 (1990), 742–751.

[KS94] S. Kirkpatrick and B. Selman, "Critical Behavior in the Satisfiability of Boolean Formulae," *Science* 264 (1994), 1297–1301.

[KST94] H. Kaplan, R. Shamir and R. E. Tarjan, "Tractability of Parameterized Completion Problems on Chordal and Interval Graphs: Minimum Fill-In and DNA Physical Mapping," in: *Proc. 35th Annual Symposium on the Foundations of Computer Science (FOCS)*, IEEE Press (1994), 780–791.

[KT92] A. Kornai and Z. Tuza, "Narrowness, Pathwidth and Their Application in Natural Language Processing," *Discrete Applied Mathematics* 36 (1992), 87–92.

[Len90] T. Lengauer, "VLSI Theory," in: *Handbook of Theoretical Computer Science vol. A*, Elsevier (1990), 835–868.

[Lev86] L. Levin, "Average Case Complete Problems," *SIAM J. Computing* 15 (1986), 285–286.

[LK73] S. Lin and B.W. Kernighan, "An Effective Heuristic Algorithm for the Traveling Salesman Problem," *Operations Research* 21 (1973), 498–516.

[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.

[LR91] M. Langston and S. Ramachandramurthi, "Dense Layouts for Series-Parallel Graphs," *Proc. Great Lakes Symposium on VLSI* (1991), 14–17.

[Luks82] E. Luks, "Isomorphism of Graphs of Bounded Valence Can Be Tested in Polynomial Time," *J. Comput. and Systems Sci.* 25 (1982), 42–65.

[Mad72] W. Mader, "Hinreichende Bedingungen fuer die Existenz von Teilgraphen, die zu einem vollstaendigen Graphen homomorph sind," *Abt. Math, Sem. Hamburg Univ.* 37 (1972), 86–97.

[MP94] S. Mahajan and J. G. Peters, "Regularity and Locality in k-Terminal Graphs," *Discrete Applied Mathematics* 54 (1994), 229–250.

[MR95a] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge Univ. Press, 1995.

[MR95b] S. Mahajan and H. Ramesh, "Derandomizing Semidefinite Programming Based on Approximation Algorithms," in *Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science* (1995), 162–169.

[MR98] M. Mahajan and V. Raman, "Parameterizing Above the Guarantee: MaxSat and MaxCut," to appear in *J. Algorithms.*

[MSL92] D. Mitchell, B. Selman and H. Levesque, "Hard and Easy Distributions of SAT Problems," in: *Proceedings AAAI-92*, AAAI (1992), 459–465.

[NP85] J. Nešetřil and S. Poljak, "On the Complexity of the Subgraph Problem," *Comm. Math. Univ. Carol.* 26 (1985), 415–419.

[Pet81] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, 1981.

[PGWRS96] C. Papadimitriou, O. Goldreich, A. Wigderson, A. Razborov and M. Sipser, "The Future of Computational Complexity Theory: Part I," *SIGACT News* 27 (1996), 6–12.

[PY96] C. Papadimitriou and M. Yannakakis, "On Limited Nondeterminism and the Complexity of the VC Dimension," *J. Computer and Systems Sciences* 53 (1996), 161–170.

[Rag88] P. Raghavan, "Probabilistic Construction of Deterministic Algorithms: Approximating Packing Integer Programs," *J. Computer and Systems Sciences* 37 (1988), 130–143.

[RS85] N. Robertson and P. D. Seymour, "Graph Minors: A Survey," in *Surveys in Combinatorics*, I. Anderson, ed. (Cambridge University Press: Cambridge, 1985), 153–171.

[RS96] N. Robertson and P. D. Seymour, "Graph Minors XV. Giant Steps," *J. Comb. Theory Ser. B* 68 (1996), 112–148.

[Sha97] R. Shamir. Private communication, 1997.

[SOWH96] D. L. Swofford, G. J. Olsen, P. J. Waddell and D. M. Hillis, "Phylogenetic Inference," in: D. M. Hillis, C. Moritz and B. K. Mable (eds.) *Molecular Systematics.* Sinauer Associates, Inc. (1996), 407–514.

[SS77] R. Solovay and V. Strassen, "A Fast Monte Carlo Test for Primality," *SIAM J. Computing* (1977), 84–85.

[Th97] M. Thorup, "Structured Programs Have Small Tree-Width and Good Register Allocation," *Proceedings 23rd International Workshop on Graph-Theoretic Concepts in Computer Science, WG'97*, R. Möhring (ed.), Springer Verlag, Lecture Notes in Computer Science vol. 1335 (1997), 318–332.

[To91] S. Toda, "PP is as Hard as the Polynomial Time Hierarchy," *SIAM J. Comput.* (1991), 865–877.

[Wilf85] H. Wilf, "Some Examples of Combinatorial Averaging," *Amer. Math. Monthly* 92 (1985), 250–261.

[Yan95] M. Yannakakis. "Perspectives on Database Theory," *Proceedings of the IEEE Symposium on the Foundations of Computer Science* (1995), 224–246.

[YV70] A. S. Yessenin-Volpin, "The Ultraintuitionistic Criticism and the Antitraditional Program for Foundations of Mathematics," in: *Intuitionism and Proof Theory: Proceedings of the Summer Conference at Buffalo, N.Y., 1968*, A. Kino, J. Myhill and R. E. Vesley (eds.), North-Holland, 1970.

Rodney G. Downey, School of Mathematics and Computing Sciences, P.O. Box 600, Victoria University, Wellington, New Zealand. Research supported by a grant from the United States/New Zealand Cooperative Science Foundation and by the New Zealand Marsden Fund for Basic Science.
E-mail address: rod.downey@vuw.ac.nz

Michael R. Fellows, Department of Computer Science, University of Victoria, Victoria, B.C. V8W 3P6, Canada. Research supported by the National Science and Engineering Research Council of Canada.
E-mail address: mfellows@csr.uvic.ca

Ulrike Stege, Computational Biochemistry Research Group, ETH Zürich, Ch-8092 Zürich, Switzerland
E-mail address: stege@inf.ethz.ch