Parameterized Complexity: New Developments and Research Frontiers

Michael R. Fellows
Department of Computer Science
University of Victoria
Victoria, B.C. Canada V8W 3P6

1. Introduction

This survey of the current research horizons and new directions in the area of parameterized complexity is loosely based on a series of three lectures given in January 2000, in Kaikoura on the South Island of New Zealand. The occasion was the remarkably pleasant annual community conference series hosted by the New Zealand Mathematical Society.

One of the principal purposes of the meeting is that of encouraging and supporting graduate students to become involved in research. There are many interesting and unexplored possibilities for graduate research topics, having real and exciting applications, as well as involving some of the deepest areas of combinatorics, in this new branch of computer science theory. Throughout the exposition we will make some effort to highlight these opportunities, and to try to keep things accessible to students and non-specialists.

2. Parameterized Complexity in a Nutshell

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

• The basic empirical motivation.

• The relationship of parameterized complexity to other contemporary research directions in theoretical computer science that address the central problem of coping with intractability.

• The perspective provided by three fundamental forms of the Halting Problem.

• The natural relationship of parameterized complexity to heuristics and practical computing strategies.

2.1. Empirical Motivation: Two Forms of Fixed-Parameter Complexity

Most natural computational problems are defined on input consisting of various information. A simple example is provided by the many graph problems that are defined as having input consisting of a graph $G = (V, E)$ and a positive integer k, such as (see [GJ79] for definitions), Graph Genus, Bandwidth, Min Cut Linear Arrangement, Independet Set, Vertex Cover and Dominating Set. The last two problems are defined

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

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

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

1. After many rounds of improvement involving a variety of clever ideas, the best known algorithm for Vertex Cover runs in time $O(1.271^k + kn)$ [CKJ99]. This algorithm has been implemented and is quite practical for n of unlimited size and k up to around 200 [HGS98, St00]. In fact, it has recently been shown that for planar graphs (where the Vertex Cover problem is still *NP*-complete), for any $\epsilon > 0$, the problem can be solved in time $O((1 + \epsilon)^k + kn)$ [Chen00].

2. The best known algorithm for Dominating Set is *still* just the brute force algorithm of trying all k-subsets. For a graph on n vertices this approach has a running time of $O(n^{k+1})$.

The table below shows the contrast between these two kinds of complexity. Note that the table just shows the competition between $2^k n$ and n^k, while the contrast between (1.) and (2.) is actually much more dramatic than this.

	$n = 50$	$n = 100$	$n = 150$
$k = 2$	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 order to formalize the difference between VERTEX COVER and DOMINATING SET we make the following basic definitions.

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 $\Sigma^* \times \mathbb{N}$.*

In particular, it makes no difference to the theory if a parameter is non-numerical; there are many natural examples for this, such as the GRAPH MINOR, GRAPH TOPOLOGICAL CONTAINMENT and SUBGRAPH ISOMORPHISM problems, where the natural parameter is the "guest" graph. A parameter can also be an aggregate of various kinds of information, such as a pair (H, Δ) representing, for example, the guest graph H and a maximum degree bound Δ on the host graph.

Definition 2.2 *A parameterized language L is* multiplicatively fixed-parameter tractable *if it can be determined in time $f(k)q(n)$ whether $(x, k) \in L$, where $|x| = n$, $q(n)$ is a polynomial in n, and f is a function (unrestricted). The family of fixed-parameter tractable parameterized languages is denoted FPT.*

Definition 2.3 *A parameterized language L is* additively fixed-parameter tractable *if it can be determined in time $f(k) + q(n, k)$ whether $(x, k) \in L$, where $|x| = n$, $q(n, k)$ is a polynomial in n and k, and f is a function (unrestricted). The family of fixed-parameter tractable parameterized languages is denoted FPT.*

Exercise. Show the rather surprising, and somewhat dramatic fact that a parameterized language is additively fixed-parameter tractable if and only if it is multiplicatively fixed-parameter tractable. This emphasizes how cleanly fixed-parameter tractability isolates the computational difficulty in the complexity contribution of the parameter.

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

• *The size of a database query.* Normally the size of the database is huge, but frequently queries are small. If n is the size of a relational database, and k is the size of the query (which of course bounds the number of variables in the query), then determining whether there are objects described in the database that have the relationship described by the query can be solved trivially in time $O(n^k)$. It is known that this problem is unlikely to be *FPT* [DFT96, PY97].

• *The nesting depth of a logical expression.* ML is a logic-based programming language for which relatively efficient compilers exist. One of the problems the compiler must solve is the checking of the compatibility of type declarations. This problem is known to be complete for *EXP* (deterministic exponential time) [HM91], so the situation appears discouraging from the standpoint of classical complexity theory. However, the implementations work well in practice because the ML TYPE CHECKING problem is *FPT* with a running time of $O(2^k n)$, where n is the size of

the program and k is the maximum nesting depth of the type declarations [LP85]. Since normally $k \leq 10$, the algorithm is clearly practical.

• *The number of sequences in a bio-informatics multiple molecular sequence alignment.* These might be the 7 kinds of human hemoglobin, or the cytochrome C sequences from a family of 20 related species, etc. Frequently this parameter is in a range of $k \leq 50$. Here n is governed by the lengths of the sequences, generally quite large. The problem can be solved in time $O(n^k)$ by dynamic programming. It is currently an **open problem** whether this problem is *FPT* for alphabets of fixed size [BDFHW95]. This of course is the form of the problem that the biologists are truly interested in.

• *The number of processors in a practical parallel processing system.* This is frequently in the range of $k \leq 64$. **Is there a practical and interesting theory of parallel FPT?** Two papers that have begun to explore this area (from quite different angles) are [CDiI97] and [DRST01].

• *The number of variables in a logical formula, or the number of steps in a deductive procedure.* Some initial studies of applications of parameterized complexity to logic programming and artificial intelligence have recently appeared [Tr01, GSS01], but much remains unexplored. Is it *FPT* to determine if k steps of resolution are enough to prove a formula unsatisfiable? **Can FPT be characterized in terms of bounded variable logics?** The parameterized complexity of constraint satisfaction problems is so far unexplored, one of the workhorses of applied artificial intelligence.

• *The number of steps for a motion planning problem.* In general, where the description of the terrain has size n (which therefore bounds the number of movement options at each step), we can solve this problem in time $O(n^{k+1})$ trivially. **Are there significant classes of motion planning problems that are fixed-parameter tractable?** Exploration of this topic has hardly begun [CW95].

• *The number of moves in a game.* The usual computational problem here is to determine if a player has a winning strategy. While most of these kinds of problems are *PSPACE*-complete classically, it is known that some are *FPT* and others are likely not to be *FPT*, when parameterized by the number of moves of a winning strategy [ADF95]. The size n of the input game description usually governs the number of possible moves at any step, so there is a trivial $O(n^k)$ algorithm that just examines the k-step game trees exhaustively. **The entire subject of the parameterized complexity of games is almost completely unexplored.** This is potentially a very fruitful area, since games are used mathematically to model many different kinds of situations.

• *The size of a substructure.* The complexity class $\#P$ (see the survey in this volume by Dominic Welsh) is concerned with whether the number of solutions to a problem (e.g., the number of Hamilton circuits in a graph, or the number of perfect matchings) can be counted in polynomial time. As pointed out by Dominic at the Kaikoura meeting, it would also be interesting to consider whether *small* substructures can be counted in *FPT* time, where the parameter is the size of the

substructure (e.g., circuits of length k, or k-matchings). This subject has only just begun to be explored [Ar00].

• *The distance from a guaranteed solution.* Mahajan and Raman pointed out that for many problems, solutions with some "intermediate" value (in terms of n) may be guaranteed and that it is then interesting to parameterized above or below the guaranteed value [MR98]. For a simple (and open) example, by the Four Color Theorem and the Pigeon Hole Principle it is always possible to find a 4-coloring of a planar graph where at least one of the colors is used at least $n/4$ times. **Is it *FPT* to determine if a planar graph admits a 4-coloring where one of the colors is used at least $n/4 + k$ times?** (Exercise: show that this can be solved in time $O(n^{O(k)})$.

• *The amount of "dirt" in the input or output for a problem.* For example, we might have an application of graph coloring where the input is expected to be 2-colorable (a problem that is in P) except that due to some imperfections, the input is actually only "nearly" 2-colorable. It would then be of interest to determine whether a graph can be properly colored in such a way that at most k vertices receive a third color. Some results indicating that the problem might be *FPT* have been given by Leizhen Cai and Baruch Scheiber [CS97].

• *The "robustness" of a solution to a problem, or the distance to a solution.* For example, given a solution of the Minimum Spanning Tree problem in an edge-weighted graph, we can ask if the cost of the solution is robust under all increases in the edge costs, where the parameter is the total amount of cost increases. A number of problems of this sort have recently been considered by Leizhen Cai [Cai01]. As a further example of a similar kind of problem, we might be given a directed graph and asked if reversing at most k arcs is sufficient to obtain strong connectivity [Ros01].

These are just a few examples to stimulate thinking. If you look around, it is obvious that the practical world is full of interesting concrete problems governed by parameters of all kinds that are bounded in some small or moderate range. If we can design algorithms with running times like $2^k n$ for these problems, then we may have something really useful. There are now many examples where we can do this for important problems that are *NP*-complete or worse.

In the classical framework, restricting the input to a problem can lead to polynomial time complexity, but generally, most hard (*NP*-complete or worse) problems remain hard when restricted to planar graphs and structures, for example. In the parameterized framework, almost all problems turn out to be *FPT* when restricted to planar inputs. In fact, for many planar parameterized graph problems, Kloks, Niedermeier and others have recently shown that *FPT* complexities of the form $c^{\sqrt{k}}n$ can be obtained [AFN01, GK01]. This immediately raises the question of whether *FPT* complexities of this form might be achievable for the general unrestricted problems (such as the general parameterized Vertex Cover problem).

Using a very clever problem parameterization of the form: *Does G have a vertex cover of size at most $k \log n$?* (just to give a very rough indication of their

approach), Liming Cai and David Juedes [CJ01] have proved the astonishing result that the general VERTEX COVER problem cannot have an FPT complexity of the form $O(2^{o(k)}p(n))$ unless the W-hierarchy collapses. This goes beyond our list above of natural problem parameters, and shows what structural complexity insights might be achieved with unnatural, mathematically engineered parameterizations.

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

Definition 2.4 *A parameterized language L belongs to the class XP (slicewise P) if it can be determined in time $f(k)n^{g(k)}$ whether $(x,k) \in L$, where $|x| = n$, α is a constant independent of both n and k, with f and g being unrestricted functions.*

Is it possible that $FPT = XP$? This is one of the few structural questions concerning parameterized complexity that currently has an answer [DFS99].

Theorem 2.5 *FPT is a proper subset of XP.*

2.2. The Main Challenge for Computational Complexity Theory Today

In the opening remarks of Eric Allender's first talk at the Kaikoura meeting he suggested that, "Complexity theory, at least to some extent, is an empirical discipline." Along these lines, it can be said that when the modern framework of complexity theory first emerged, with the beautiful and productive twin ideas of:

1. Polynomial time
2. *NP*-completeness

it could not have been anticipated how the natural world of computational problems would sort out on this basic axis of complexity classification. Thousands of theorems later, we can speak with confidence about the situation that has been revealed in this mathematico-empirical way.

"Almost everything is *NP*-complete or worse!"

The natural world of computational problems is, generally speaking, relatively hostile to efficient forms of information processing, by the evidence that has accumulated in the classical complexity framework. This basic empirical pattern of scientific discovery continues with undiminished strength, especially as the ideas of computational complexity continue to permeate through all of the sciences. Major results in computational complexity in the last few years include:

- The Ising model in three or more dimensions is *NP*-complete [Ist00]. This is essentially a major result in theoretical physics, because of the importance of the Ising model in the study of phase transitions.

- The protein-folding problem is *NP*-hard in two or more dimensions [BL97, CGPPY98]. Understanding protein folding is one of the Holy Grail problems of biochemistry.

- Many basic computational problems of interest to political science, sociology and economics are *NP*-hard.

One of the most fundamental challenges for theoretical computer science that has emerged from this initial period of empirical discovery is:

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

The reader should consult Chapter 6 of [GJ79], "Coping with *NP*-completeness," for a foundational discussion of this issue. One of the first things that was noticed was that for problems that have numerical input, it can make a crucial difference whether this information is presented in unary or in binary. If a problem remains *NP*-complete when the numerical information is in unary, then the problem is termed *strongly NP-complete*. Moshe Vardi was perhaps the first to discuss the importance of the different ways that different aspects of the input may contribute to overall problem complexity in really fundamental ways, in the context of database theory [Var82] (see also [VW86]). In database query problems the database is normally *huge* and the query, by comparison, quite small — so it is *essential* to clarify the qualitative nature of the complexity contributions of these different aspects of the input — the main proposition of [Var82]. This seminal discussion was eventually further developed by Papadimitriou and Yannakakis (after the issue was posed by Yannakakis at STOC in 1995 [Yan95]), in the influential paper [PY97] (see also [DFT96]). All of this discussion of the contributions of input structure to overall complexity in the context of classic database problems has been carried even further in the recent work of Martin Grohe and coworkers, who have studied the issues for databases of bounded treewidth [GM98, GSS00].

The main research programs that have so far been articulated to rise to the central challenge of coping with intractability make a rather short list.

- **Average-case analysis.** The idea here is that we might be able to show that many hard problems can be solved by algorithms that run in polynomial-time *on average* for expected input distributions.

- **Approximation algorithms.** The idea here is that we might be able to find polynomial-time algorithms that do well for hard problems by finding solutions that are approximately optimal.

- **Randomized algorithms.** The idea here is that we might be able to beat intractability by employing randomized polynomial-time algorithms. These might be able to solve the problem in time that is expected to be polynomial, regardless of the input disribution.

- **DNA and quantum computing.** The hope here is that fundamentally more powerful computing devices will empower us to make an effective attack on classical computational intractability.

- **Improved worst-case exponential algorithms.** In this program, we simply focus on designing algorithms with improved exponential running times.

- **Heuristics and meta-heuristics.** This includes a variety of approaches that often involve some mathematical sophistication, but that frequently appeal to various biological or physical metaphors for understanding and coping with computational intractablity: *simulated annealing*, *genetic algorithms*, *neural nets*, *roaming ants*, *memetic algorithms*, etc.

- **Parameterized complexity analysis and algorithm design.** This of course is our topic. The basic idea is to try to confine the "inevitable" combinatorial explosion to some aspect of the input (the parameter) that might be expected to be small for realistic input distributions.

Of these research programs, parameterized complexity is by far the least well known and the least explored. It is also somewhat orthogonal to these other programs, and intersects with them in ways that are potentially productive. Much of the current excitement in the area of parameterized complexity is concerned with these intersections, some of which are described below. Of course, at the end of the day, all of these approaches contribute to understanding and coping with computational intractability.

Intersection 1. Research on quantum computing currently revolves around the central notion of *quantum polynomial time* QP. **What parameterized problems are** FPT **for quantum computation?** The quantum computers that are built in the next few decades seem likely to operate on a limited number of quantum bits, *qubits*, which is therefore a natural parameter to consider as it interacts with problem complexities. Perhaps there is a way to use quantum computing to make bounded treewidth algorithmics practical. Quantum computation seems to pertain best to problems with algebraic structure. Since bounded treewidth algorithms can be cast in finite-state terms, and finite automata support various algebraic decompositions (e.g., via Krohn-Rhodes Theorem), there may be grounds for hope in this direction. If FPT makes sense, then so does $QFPT$, especially since the number of qubits might align, as an engineering parameter, with the algorithmic parameter considered.

Intersection 2. Echoing the first example, research on randomized algorithms has focused on the notion of randomized polynomial-time. In fact, randomized algorithms have many applications for the few problems that are in P and are something of a practical success story, because they are frequently simpler to program. **Randomized** FPT **is almost completely unexplored**.

Intersection 3. The emphasis in the area of approximation algorithms is on the notions of:

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

The connections between the *parameterized complexity* and *polynomial-time approximation* programs are actually very deep and developing rapidly. One of the reasons is that as one considers approximation schemes, there is immediately a parameter staring you in the eye: *the goodness of the approximation*. To illustrate what can happen, the first P-time approximation scheme for the Euclidean TSP due to Arora [Ar96], gave solutions within a factor of $(1+\epsilon)$ of optimal in time $O(n^{35/\epsilon})$. **Can we get the $k = 1/\epsilon$ out of the exponent?** is a concrete question that calls out for further clarification for many known P-time approximation schemes. The following definition captures the essential issue.

Definition 2.6 *An optimization problem* Π *has an* efficient P-time approximation scheme *if it can be approximated to a goodness of* $(1+\epsilon)$ *of optimal in time* $f(k)n^c$ *where* c *is a constant and* $k = 1/\epsilon$.

A theorem giving some general information about this issue has been proved by Cristina Bazgan (independently by Cesati and Trevisan) [Baz95, CT97].

Theorem 2.7 *Suppose that* Π_{opt} *is an optimization problem, and that* Π_{param} *is the corresponding parameterized problem, where the parameter is the value of an optimal solution. Then* Π_{param} *is fixed-parameter tractable if* Π_{opt} *has an efficient PTAS.*

The theorem has powerful applications contrapositively, showing limits to approximation when we can demonstrate that problems are unlikely to be *FPT*.

Intersection 4. A landmark result in the program of improved worst-case exponential algorithms for *NP*-hard problems is the algorithm of Robson [Rob86] that computes a maximum independent set (MIS) in a graph in time $O(1.212^n)$. This is a significant improvement on the obvious $O(2^n)$ algorithm of trying all subsets of the vertex set. We can make the following interesting series of reflections:

- In $G = (V, E)$ a subset $V' \subseteq V$ is an independent set if and only if $V - V'$ is a vertex cover. The Vertex Cover problem is fixed-parameter tractable, with the currently best known algorithm (due to Chen, Kajn and Jia) having a running time of $(1.271^k + kn)$ [CKJ99]. Thus we have the following alternative way of computing a MIS: use the parameterized vertex cover algorithm to find the smallest possible vertex cover, and take the complement. Since we have no advance knowledge of k, which might be as large as n, this approach has a running time of $O(1.271^n)$, and is apparently *not* an improvement on Robson — but "almost".

- However, if the expected size of the MIS is around $n/2$, then the Chen-Kajn-Jia algorithm will give us a running time of $O((\sqrt{1.271})^n) = O((1.12)^n)$, and now it looks like the approach via the *FPT* algorithm for the "dual" problem is actually superior.
- On the other hand, this causes one to wonder whether the Robson algorithm can be modified to be sensitive ("output sensitive") to the size of the MIS that it finds. The question is: Can we adapt the Robson algorithm so that it runs in time $O((1.212)^k n)$ where k is the size of the MIS that it produces? (Note that we are asking now if Robson's algorithm can be adapted into an *FPT* algorithm!) If so, then Robson's algorithm would again be superior when the expected size of the MIS is around $n/2$.
- This is unlikely, because the Independent Set problem, parameterized by the size of the independent set, is hard for $W[1]$ and therefore very probably *not* in *FPT* [CCDF96]. The reader can find more about $W[1]$-hardness (parametric intractability) in §2.3. Close inspection of the Robson algorithm also shows that it is "designed around n" and does not seem to easily admit any kind of modification sensitive to the size of the MIS computed. The route via Vertex Cover is a *pay for what you get* approach. Robson's algorithm, by contrast, pays according to the size n of the graph, because it is fundamentally insensitive to the extra structure afforded by parameterization.

It is clear that there is a rich intersection of these two research programs. It is worth noting in passing that many problems have a natural dual form such as we have exploited here (the k-MIS problem is the same as the $(n-k)$-Vertex Cover problem), and it is "almost" a general rule, first noted by Raman, that parametric duals of *NP*-hard problems have complementary parameterized complexity (one is *FPT*, and the other is $W[1]$-hard) [KR00]. For example, determining whether a graph has a dominating set of size $n-k$ is *FPT*, as is determining whether a graph can be colored with $n-k$ colors. The analogous problem for Bandwidth is still open. For a catalog of examples, see [AFMRRRS00].

Intersection 5. Local search is a mainstay of heuristic algorithm design [AL97]. The basic idea is that one maintains a *current solution*, and iterates the process of moving to a neighboring "better" solution. A neighboring solution is usually defined as one that is a single step away according to some small edit operation between solutions. The following problem is completely general for these situations, and could potentially provide a valuable subroutine for "speeding up" local search:

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

There are many practical algorithms based on local search. **When is the k-Speed Up problem *FPT*?** Using the methods of [PV91] or [AYZ94], it can

be shown that the problem of determining whether an impoverished travelling salesman can visit at k cities and return home for a given budget is in *FPT*. Can this *FPT* algorithm for the SHORT CHEAP TOUR problem be used as a building block for a new TSP heuristic?

Much recent work in the design of local search heuristics has explored hybrid strategies of local search and genetic algorithms that maintain a population of solutions and combine these in various ways to attempt to discover improved solutions. In the memetic paradigm, Pablo Moscato has recently studied natural parameters that describe the recombination process [Mo01]. For example, in a population of good solutions, one might try to identify k features common to all of them, and then reduce the size of the input by locking these in.

We will say more about the connections of parameterized complexity to systematic heuristics, meta-heuristics and practical computing in general in §2.4.

2.3. The Halting Problem: A Central Reference Point

The main investigations of computability and efficient computability can be classified according to three basic forms of the Halting Problem that anchor the discussions.

2.3.1. Is there any algorithm? The basic form of the HALTING PROBLEM is defined:
THE HALTING PROBLEM
Input: A Turing machine M.
Question: If M is started on an empty input tape, will it ever halt?

In other words, equivalently, if someone gives you the (finite) text or computer code for an algorithm, is there a way to analyze this text to determine whether or not it will go into an infinite loop if executed? The answer is no. By a slight modification of Cantor's diagonalization argument, we have what was historically one of the first big pieces of bad news about computing: there is no algorithm to solve the Halting Problem. If that were the end of the story, this might be bad news for people who build operating systems and nobody else would be bothered.

However, many combinatorial reductions were found that reduced the HALTING PROBLEM to some other problem Π. That is, it was shown that if Π had an algorithm, then so would the HALTING PROBLEM. Thus developed a small empirical avalanche of bad news as mathematical investigation of the natural landscape of computing continued. Dozens of important, natural, and in some cases rather harmless looking problems are now provably known not to have any algorithm whatsoever. It is also worth noting that Gödel's Incompleteness Theorem — bad news about the mechanization of mathematical proof — is an easy corollary to the unsolvability of the HALTING PROBLEM.

2.3.2. Is there a polynomial-time algorithm (like for SORTING?) The second important form of the HALTING PROBLEM is the one that sets up the P versus NP discussion:

THE POLYNOMIAL-TIME HALTING PROBLEM FOR NONDETERMINISTIC TURING MACHINES
Input: A nondeterministic Turing machine M.
Question: Is it possible for M to reach a halting state in n steps, where n is the length of the description of M?

This problem is trivially *NP*-complete, and in fact essentially defines the complexity class *NP*, the class of decision problems that can be solved by nondeterministic polynomial-time Turing machines. For a concrete example of why it is trivially *NP*-complete, consider the 3-COLORING problem for graphs, and notice how easily it reduces to the P-TIME NDTM HALTING PROBLEM. Given a graph G for which 3-colorability is to be determined, I just create the following nondeterministic algorithm:

Phase 1. (There are n lines of code here if G has n vertices.)
(1.1) Color vertex 1 one of the three colors nondeterministically.
(1.2) Color vertex 2 one of the three colors nondeterministically.
...
(1.n) Color vertex n one of the three colors nondeterministically.

Phase 2. Check to see if the coloring is proper and if so halt. Otherwise go into an infinite loop.

It is easy to see that the above nondeterministic algorithm has the possibility of halting in n steps if and only if the graph G admits a 3-coloring.

Reducing any other problem $\Pi \in NP$ to the P-TIME NDTM HALTING PROBLEM is no more difficult than taking an argument that the problem Π belongs to NP and modifying it slightly to be a reduction to this form of the HALTING PROBLEM. It is in this sense that the P-TIME NDTM HALTING PROBLEM is essentially the *defining* problem for NP.

GRAPH 3-COLORING is an elegant combinatorial problem. In contrast, the P-TIME NDTM HALTING PROBLEM is sort of ugly and disgusting. We are confronted with an amorphous, opaque, mixed mass of nondeterministic computational possibilities — unstructured programming gone beserk, and we are asked to analyze this code with respect to a safety issue: can it possibly halt in $q(n)$ steps? This is worse than software engineering!

The fact that it is so "ugly" and unanalyzable is what makes this problem important. Because we have this strong intuitive feeling that there is little we can do efficiently to analyze a nondeterministic Turing machine for its halting behaviour, it is reasonable to make the following conjecture.

Conjecture 1. There is no polynomial-time algorithm to solve the P-TIME NDTM HALTING PROBLEM. That is, $P \neq NP$.

Most computer scientists find this conjecture compelling. It is widely considered to be the most important unsolved problem in mathematics and computer science. So much for what we probably can't do! But trivially, we *can* solve

the P-TIME NDTM HALTING PROBLEM in exponential time $O(n^{p(n)})$, where p is a polynomial in n, by exploring all possible computation paths of length n, and seeing if any of them lead to a halting state. The P-TIME NDTM HALTING PROBLEM is thus a generic computational embodiment of exponential search. The issue, relative to Conjecture 1, is whether we can get the polynomial $p(n)$ in the trivial exhaustive $O(n^{p(n)})$ algorithm for the problem *out of the exponent* and solve the problem in polynomial time. For this ugly, opaque, seemingly structureless problem, most of us reasonably conjecture that this will not be possible.

Starting from this reference point, over time, a huge avalance of bad news has empirically accumulated. The shocking thing is that there is an immense wealth of combinatorial reductions from the ugly P-TIME NDTM HALTING PROBLEM, (many of these passing initially through SATISFIABILITY — the importance of Cook's Theorem), to elegant combinatorial problems such as GRAPH 3-COLORING. We now know that thousands of natural problems that can be solved by polynomial-in-the-exponent algorithms are no more likely to be solvable by polynomial-time (not-in-the-exponent) algorithms than the P-TIME NDTM HALTING PROBLEM.

2.3.3. Is there an *FPT* algorithm (like for VERTEX COVER)? Interest in algorithms and complexity ranges far and wide, and will continue to do so as all of the Sciences, as well as Linguistics, Anthropology, Classics, Forestry, Art History, Music, etc. — everything — generate oceans of data with the new tools for information collection and generation. Most "foreigners" in these other fields now have a nodding familiarity with computing.

We can interpret the dialog in which Theoretical Computer Science stands with these other fields as consisting currently of requests for polynomial-time algorithms. They come looking for efficient algorithms, and we (the complexity theorists and algorithm designers) are prepared to answer them when they ask, "May I please have an algorithm for my problem that is like Bubblesort (at least)?"

The dialog will inevitably evolve. The Biologist who has been informed that PROTEIN FOLDING is *NP*-complete will naturally ask, "Well then, seeing as the critical folding domains involve about $k = 70$ contact points between hydrophobic molecules — can I please have an algorithm for my problem that is like VERTEX COVER?"

The following fundamental flavor of the HALTING PROBLEM necessarily anchors the subsequent discussion.

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

This can be trivially solved in time $O(n^k)$ by exploring the depth k, n-branching tree of possible computation paths exhaustively.

Conjecture 2. There is no FPT algorithm to solve this problem (in technical terms, $FPT \neq W[1]$).

Our intuitive evidence for this conjecture is essentially the same as for Conjecture 1. We do not expect to be able to get the parameter k out of the exponent. We do not expect to be able to solve this problem in time $f(k) + n^c$ like VERTEX COVER. In fact, it seems quite difficult to imagine solving the problem in time $O(n^9)$ when $k = 10$. One could reasonably maintain that our intuitions about Conjecture 1 are exposed in Conjecture 2 with even more compelling directness, although technically Conjecture 2 is stronger (Conjecture 2 implies $P \neq NP$, but the reverse implication is not known to hold).

The k-STEP NDTM HALTING PROBLEM is complete for the parameterized complexity class $W[1]$, which is therefore a strong analog of NP. We can now clarify a little of what was said earlier about various problems being unlikely to be fixed-parameter tractable. The DOMINATING SET problem is unlikely to be in FPT because a reduction has been found from the k-STEP NDTM HALTING PROBLEM to DOMINATING SET, which means that the DOMINATING SET problem cannot be in FPT unless the same is true for the k-STEP NDTM HALTING PROBLEM. The following definition tells what we mean by this kind of reducibility.

Definition 2.8 *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.*

Exercise. Show that if L and L' are parameterized languages and L reduces to L', then $L' \in FPT$ implies $L \in FPT$.

Parameterized languages L and L' are *equivalent* in parametric complexity if L can be reduced to L' and vice versa. An equivalence class under this notion of equivalence is a *degree* of parametric complexity. The complexity class $W[1]$ (the parametric analog of *NP*) is the degree of the k-STEP NDTM HALTING PROBLEM. The main degree sequence of parameterized complexity is

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

It is conjectured that all of these containments are proper, but all that is currently known is that FPT is a proper subset of XP. We won't go any further into the theory of parametric intractability than this. **There are only the barest beginnings of a structure theory of parametric intractability.**

Area for investigation. There is a "lemonade" that is made from the "lemons" of intractability: cryptography! The notion of parametric intractability should

therefore provide the basis for parameterized cryptosystems whose security guarantees rest on plausible conjectures that the parameterized cracking problems are not FPT, even though for fixed values of the parameter k the k-parameterized cryptosystem can be cracked in polynomial time (with k in the exponent).

Area for investigation. We would like to know if there is an FPT algorithm that for input G and k (the parameter) will compute a dominating set for G whose size is within a factor of $(1+\epsilon)$ of optimal, where $\epsilon = 1/k$. In fact, this is the most natural way in which to parameterize the complexity of approximation — this is the *canonically parameterized* approximation problem for the MINIMUM DOMINATING SET problem. Note that essentially we are asking if MINIMUM DOMINATING SET has an efficient PTAS. However, we already know the answer to this. By Bazgan's Theorem, since DOMINATING SET is $W[2]$-complete, we cannot have an FPT algorithm for this canonical approximation problem unless $FPT = W[2]$. But now consider the analogous canonical parameterized problem of approximation for VERTEX COVER. We cannot apply Bazgan's Theorem, but we still know the answer! The canonical parameterized approximation problem for VERTEX COVER is hard for $W[P]$. We know this because it has been shown that there is no PTAS (and therefore no EPTAS) for VERTEX COVER unless $P = NP$ which in turn implies $FPT = W[P]$ (and this is what we mean by "hard for $W[P]$"). We know this only by the daunting machinery of holographic proof systems. The area for investigation is this: can we directly and combinatorially encode $W[1]$ into the canonical parameterized approximation problem for VERTEX COVER, thus greatly simplifying and perhaps sharpening the study of P-time approximability?

2.4. Connections to Practical Computing and Heuristics

What is practical computing, anyway? An amusing and thought-provoking account of this issue has been given by Karsten Weihe in the paper, "On the Differences Between Practical and Applied," [Wei00].

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

In considering stories of practical computing, we are quickly forced to give up the comfortable and traditional myths that these fill up the definitional spaces of our mathematical modeling — or are random according to some distribution.

An interesting example is given by Weihe of a problem concerning the train systems of Europe. Consider a bipartite graph $G = (V, E)$ where V is bipartitioned into two sets S (stations) and T (trains), and where an edge represents that a train t stops at a station s. The relevant graphs are huge, on the order of 10,000 vertices. The problem is to compute a minimum number of stations $S' \subseteq S$ such that every train stops at a station in S'. It is easy to see that this is a special case of the HITTING SET problem, and is therefore NP-complete. Moreover, it is also

$W[1]$-hard, and so the straightforward application of the parameterized complexity program fails as well.

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

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

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

What can we learn from this example, and how does it relate to parameterized complexity? First of all, it displays one of the most universally applicable coping strategies for hard problems: *smart preprocessing*. In a mathematically precise sense, this is exactly what fixed-parameter tractability is all about. The following provides an equivalent definition of FPT that displays this connection [DFS99].

Definition 2.9 *A parameterized language 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 2.10 *A parameterized language L is fixed-parameter tractable if and only if it is kernelizable.*

As an aside, finding natural, polynomial-time kernelization algorithms for FPT problems yielding small problem kernels (e.g., $|x'| \leq ck$) turns out to be intimately related to polynomial-time approximation algorithms (e.g., to within a factor of c of optimal). This important connection between parameterized and classical complexity theory, essentially an export bridge from the former to the latter, was first pointed out in [NSS98]. See also [HL01, FMcRS01, Bellairs01] for further recent examples.

Research Topic: Systematize the derivation of polynomial-time approximation algorithms from FPT algorithms, and explore how W-hardness can detect limits to approximation.

Weihe's example looks like an *FPT* kernelization, but what is the parameter? As a thought experiment, let us define the *Karsten parameter* $K(G)$ of a graph G to be the maximum size of a component of G when G is reduced according to the two simple reduction rules above. Then it is clear, although it might seem artificial, that Hitting Set, Minimum Dominating Set and no doubt many other problems, can be solved optimally in *FPT* time for the parameter $k = K(G)$. We can add this new tractable parameterization of Minimum Dominating Set to the already known fact that Minimum Dominating Set can be solved optimally for the parameter *treewidth*. In fact, many *NP*-hard problems can be solved optimally for bounded treewidth, and treewidth is turning out to be an almost universally relevant parameter in computing.

We are seeing here an example of a problem where the natural input distribution (graphs of train systems) occupies a limited parameter range, but the relevant parameter is not at all obvious. This can also perhaps serve as an example of how reduction rules can be used in a discovery process for relevant hidden parameters of input distributions.

Research Topic: Systematize how to discover relevant hidden parameters in order to apply the full force of parameterized methods in the design of practical algorithms.

It is reasonable to suspect that the trains problem example represents a very general situation. The inputs to a computational problem are frequently the *outputs* of some other computational process (e.g., the designing and operating of train systems) that are governed by their own feasibility constraints (not only computational), and that the correct point of view is that the natural world of computing is engaged in a vast ecology of hidden parameters of feasibility. To quote from an earlier survey paper on this point [DFS99]:

> 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.

This point of view leads to the following interesting and relatively unexplored research program, first suggested by Leizhen Cai [Cai01]. The program is to understand (in the sense of *FPT versus* $W[1]$-hard) how every input-governing problem-parameter affects the complexity of every other problem. As a small example of this program for graph problems, we can make the following table. We use here the shorthand: TW is Treewidth, BW is Bandwidth, VC is Vertex

Cover, DS is Dominating Set and G is Genus. The entry in the 2nd row and 4th column indicates that there is an *FPT* algorithm to optimally solve the Dominating Set problem for a graph G of bandwidth at most k (given with a witness ordering of the vertices). The entry in the 4th row and second column indicates that it is unknown whether Bandwidth can be solved optimally by an *FPT* algorithm, where the parameter is the domination number, and the input is supplied with a minimum dominating set.

	TW	BW	VC	DS	G
TW	*FPT*	W-hard	*FPT*	*FPT*	?
BW	*FPT*	W-hard	*FPT*	*FPT*	?
VC	*FPT*	?	*FPT*	*FPT*	?
DS	?	?	W-hard	W-hard	?
G	W-hard	W-hard	W-hard	W-hard	*FPT*

Table 2: The Complex Ecology of Parameters

Our attention so far has mostly been concerned with the diagonal — Treewidth is *FPT* and Bandwidth is W-hard — as stand-alone problems. But if the natural world of complexity "runs" on a commerce of hidden parameters, as Karsten Weihe's account of the train problem suggests, then it is important to understand how different parameters interact.

3. Summary

Parameterized complexity has originated and thrived on concrete annoying problems in computational complexity. The parametric complexity of the following well-known problems is still unresolved. All three are known to belong to XP.

- Directed Feedback Vertex Set: the problem of finding k vertices in a directed graph such that every directed cycle includes on of these. (Conjectured to be *FPT* in general, yet the problem is still open for planar graphs.)
- Graph Topological Containment: the problem of determining whether a graph G has a subgraph that is a subdivision of a parameter graph H. (Conjectured to be *FPT*.)
- Longest Common Subsequence for k sequences over an alphabet of size 4. (Conjectured to be W-hard.)

We have described here the main motivations, the few main definitions, and many of the current open problems and areas of research opportunity, that pertain to exploring computational complexity in the parameterized complexity framework. The reader who wishes for a more comprehensive and detailed introduction should turn to the moderately priced and entertaining book [DF98].

References

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

[AFN01] J. Alber, H. Fernau and R. Niedermeier, "Parameterized Complexity: Exponential Speed-Up for Planar Graph Problems," manuscript 2001.

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

[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.

[Ar00] V. Arvind, "On the Parameterized Complexity of Counting Problems," *Workshop on Parameterized Complexity*, Madras, India, Dec. 2000.

[AFMRRRS00] V. Arvind, M. R. Fellows, M. Mahajan, V. Raman, S. S. Rao, F. A. Rosamond, C. R. Subramanian, "Parametric Duality and Fixed Parameter Tractability", manuscript 2000.

[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.

[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.

[Bellairs01] V. Dujmovic, M. Fellows, M. Hallett, M. Kitching, G. Liotta, C. McCartin, N. Nishimura, P. Ragde, F. Rosamond, M. Suderman, S. Whitesides and D. Wood, "A Fixed-Parameter Tractability Approach to Two-Layered Graph Drawing," manuscript, 2001, following the *International Workshop on Fixed-Parameter Tractability in Graph Drawing*, Bellairs Research Institute of McGill University, Holetown, Barbados, Feb 2001.

[BL97] B. Berger and T. Leighton, manuscript submitted to *J. Molecular Biology*, July 1997.

[Cai01] Leizhen Cai, "The Complexity of Coloring Parameterized Graphs," to appear in *Discrete Applied Math.*

[CCDF96] Liming 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.

[CDiI97] M. Cesati and M. Di Ianni, "Parameterized Parallel Complexity," Technical Report ECCC97-06, University of Trier, 1997.

[CGPPY98] P. Crescenzi, D. Goldman, C. Papadimitriou, A. Piccolboni and M. Yannakakis, "On the Complexity of Protein Folding," *Proceedings of RECOMB '98.*

[CJ01] Liming Cai and D. Juedes, "Subexponential Parameterized Algorithms Collapse the W-Hierarchy," manuscript, 2001.

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

[Chen00] J. Chen, "Simpler Computation and Deeper Theory," presentation at the *Workshop on Parameterized Complexity*, Madras, India, Dec. 2000.

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

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

[CW95] M. Cesati and H. T. Wareham, "Parameterized Complexity Analysis in Robot Motion Planning," *Proceedings 25th IEEE Intl. Conf. on Systems, Man and Cybernetics*.

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

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

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

[DRST01] F. Dehne, A. Rau-Chaplin, U. Stege and P. Taillon, "Coarse-Grained Parallel Fixed-Parameter Tractable Algorithms," manuscript, 2001.

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

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

[GK01] G. Gutin and T. Kloks, "Kernels in Planar Digraphs," manuscript, 2001.

[GM98] M. Grohe and J. Marino, "Definability and Descriptive Complexity on Databases of Bounded Treewidth," manuscript, 1998.

[GSS00] M. Grohe, T. Schwentick and L. Segoufin, "When is the Evaluation of Conjunctive Queries Tractable?" manuscript, 2000.

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

Intelligence (1999), 1–18.

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

[HL01] M. Hallett and J. Lagergren, "Efficient Algorithms for Horizontal Gene Transfer Problems," RECOMB 2001, Montreal, April 2001.

[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.

[Ist00] S. Istrail, "Statistical Mechanics, Three-Dimensionality and NP-Completeness," *Proceedings of the 32nd Annual ACM Symposium on the Theory of Computing* (STOC'00), 87-96.

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

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

[Mo01] P. Moscato, "Controllability, Parameterized Complexity, and the Systematic Design of Evolutionary Algorithms," manuscript, 2001 (http://www.densis.fee.unicamp.br/m̃oscato).

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

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

[PV91] J. Plehn and B. Voigt, "Finding Minimally Weighted Subgraphs," in *Proc. 16th International Workshop on Graph-Theoretic Methods in Computer Science*, Lecture Notes in Computer Science 484 (1991), 18–29.

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

[Rob86] J. M. Robson, "Algorithms for Maximal Independent Sets," *Journal of Algorithms* 7 (1986), 425–440.

[Ros01] F. Rosamond, *Barbados Workshop on Parameterized Complexity and Graph Drawing*, McGill University Bellairs Research Station, Holetown, Barbados, Feb. 2001.

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

[Tr01] M. Truszczynski, "On Computing Large and Small Stable Models," to appear in *Journal of Logic Programming.*

[Var82] M. Y. Vardi, "The Complexity of Relational Query Languages," *Proc. 14th ACM Symp. on Theory of Computing*, San Francisco, May 1982, ACM Press (1982), 137–146.

[VW86] M. Y. Vardi and P. Wolper, "An Automata-Theoretic Approach to Automatic Program Verification," *Proc. IEEE Symposium on Logic in Computer Science*, Boston, 1986, IEEE Press (1986), 332–344.

[Wei00] K. Weihe, "On the Differences Between Practical and Applied," Dagstuhl Workshop on Experimental Algorithmics, September 2000.

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