URL: http://www.elsevier.nl/locate/entcs/volume61.html 19 pages

Parameterized Complexity: The Main Ideas and Connections to Practical Computing

Michael R. Fellows [1]

School of Electrical Engineering and Computer Science
University of Newcastle, University Drive, Callaghan 2308, Australia

1 Introduction

Research in the parameterized framework of complexity analysis, and in the corresponding toolkit of algorithm design methods has been expanding rapidly in recent years. This has led to a flurry of recent surveys, all of which are good sources of introductory material [Ra97,Nie98,DF99,DFS99,AGN01,Gr01a,Gr01b]. One could also turn to the monograph [DF98]. Experience with implementations of *FPT* algorithms is described in [HGS98,St00,AGN01]. In several cases, these implementations now provide the best available algorithms for well-known *NP*-hard problems.

The first part of this survey attempts to summarize the main ideas of parameterized complexity and put the whole program in perspective. The second part of the survey is concerned with describing bridges to heuristics and practical computing strategies, and polynomial-time approximation algorithms.

2 Parameterized Complexity in a Nutshell

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

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

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,

[1] Email: mfellows@cs.newcastle.edu.au

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.

(i) 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 400 [HGS98,St00,DRST01].

(ii) 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.

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

A parameter may be non-numerical, and it can also be an aggregate of various kinds of information.

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).

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).

As an exercise, the reader might wish to show that a parameterized language is additively fixed-parameter tractable if and only if it is multiplicatively fixed-parameter tractable. This emphasizes how cleanly fixed-parameter tractability isolates the computational difficulty in the complexity contribution of the parameter.

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

• *The size of a database query.* Normally the size of the database is huge, but frequently queries are small. If n is the size of a relational database, and k is the size of the query, then answering the query 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 compilers work reasonably well. One of the problems the compiler must solve is the checking of the compatibility of type declarations. This problem is complete for deterministic exponential time [HM91], so the situation appears dire from the standpoint of classical complexity theory. The implementations work well in practice, using an algorithm that previously would have been called a heuristic because — we can now say — the ML TYPE CHECKING problem is solved by an *FPT* algorithm with a running time of $O(2^k n)$, where n is the size of the program and k is the maximum nesting depth of the type declarations [LP85]. Since normally $k \leq 10$, the algorithm is clearly practical.

• *The number of sequences in a bio-informatics multiple molecular sequence alignment.* Frequently this parameter is in a range of $k \leq 50$. 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].

• *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 recent 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?

• *The number of steps for a motion planning problem.* Where the description

of the terrain has size n (which therefore bounds the number of movement options at each step), we can solve this problem in time $O(n^{k+1})$ trivially. Are there significant classes of motion planning problems that are fixed-parameter tractable? Exploration of this topic has hardly begun [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. This is potentially a very fruitful area, since games are used to model many different kinds of situations.

• *The size of a substructure.* The complexity class $\#P$ 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. It would be interesting to consider whether *small* substructures can be counted (or generated) 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,Fe01].

• *A "dual" parameter.* A graph has an independent set of size k if and only if it has a vertex cover of size $n-k$. Many problems have such a natural dual form and it is "almost" a general rule, first noted by Raman, that parametric duals of *NP*-hard problems have complementary parameterized complexity (one is *FPT*, and the other is $W[1]$-hard) [KR00,AFMRRRS01]. For example, $n-k$ Dominating Set is *FPT*, as is $n-k$ Graph Coloring.

• *An unrelated parameter.* For reasons of the world, we might be presented with an input graph G together with a small set of vertices that is a dominating set in G, and be required to compute an optimal bandwidth layout. Whether this problem is *FPT* is open. Problems of this sort have recently begun to receive attention [Cai01] in the parameterized complexity literature. The relevance of hidden structure to computational complexity is discussed further in §2.

• *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 [MR99]. For a simple (and open) example, by the Four Color Theorem a planar graph must have an independent set of size at least $n/4$. Is it *FPT* to determine if a planar graph has an independent set of size at least $n/4+k$?

• *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, 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 indicate that the problem might be *FPT* [CS97], but this remains an open problem.

• *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].

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

k-SPEED UP FOR LOCAL SEARCH
Input: A solution S, k.
Parameter: k
Output: The best solution S' that is within k edit operations of S.

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

• *The goodness of an approximation.* Perhaps the single most important strategy for "coping with *NP*-completeness" [GJ79] is the program of polynomial-time approximation. The goodness of the approximation is an immediately relevant parameter. More about this in §3.

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

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

Definition 2.4 A parameterized language L belongs to the class *XP* (slice-wise *P*) if it can be determined in time $f(k)n^{g(k)}$ whether $(x,k) \in L$, where $|x| = n$, 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 [DF98].

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

2.2 The Halting Problem: A Central Reference Point

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

(i) The Halting Problem
Input: A Turing machine M.
Question: If M is started on an empty input tape, will it ever halt?

(ii) 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?

(iii) The k-Step Halting Problem for Nondeterministic Turing Machines
Input: A nondeterministic Turing machine M and a positive integer k. (The number of transitions that might be made at any step of the computation is unbounded, and the alphabet size is also unrestricted.)
Parameter: k
Question: Is it possible for M to reach a halting state in at most k steps?

The first form of the Halting Problem is useful for studying the question:

"Is there ANY algorithm for my problem?"

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

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

The second form of the Halting Problem is trivially *NP*-complete, and in fact essentially defines the complexity class *NP*. For a concrete example of why it is trivially *NP*-complete, consider the 3-Coloring problem for graphs, and notice how easily it reduces to the P-Time NDTM Halting Problem. Given a graph G for which 3-colorability is to be determined, 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 m steps (for a suitably padded Turing machine description of size m) if and only if the graph G admits a 3-coloring. Reducing any other problem $\Pi \in NP$ to the P-Time NDTM Halting Problem is no more difficult

than taking an argument that the problem Π belongs to NP and modifying it slightly to be a reduction to this form of the HALTING PROBLEM. It is in this sense that the P-TIME NDTM HALTING PROBLEM is essentially the *defining* problem for NP.

The conjecture that $P \neq NP$ is intuitively very well-founded. The second form of the HALTING PROBLEM would seem to require exponential time because there is seemingly little we can do to analyze unstructured nondeterminism other than to exhaustively explore the possible computation paths. Apart from 20 years of accumulated habit, this concrete intuition is the fundamental reference point for classical complexity theory.

When the question is:

"Is there an algorithm for my problem ... like the one for VERTEX COVER?"

the third form of the HALTING PROBLEM anchors the discussion. This question will increasingly and inevitably be asked for any *NP*-hard problem for which small parameter ranges are important in applications. It is trivially solvable in time $O(n^k)$ by exploring the n-branching, depth-k tree of possible computation paths exhaustively, and our intuition here is essentially the same as for the second form of the Halting Problem — that this cannot be improved.

The third form of the Halting Problem defines the parameterized complexity class $W[1]$. Thus $W[1]$ is very strongly analogous to NP, and the conjecture that $FPT \neq W[1]$ is very much as reasonable as the conjecture that $P \neq NP$. The appropriate notion of reduction is as follows.

Definition 2.6 A *parametric transformation* from a parameterized language L to a parameterized language L' is an algorithm that computes from input consisting of a pair (x, k), a pair (x', k') such that:

(i) $(x, k) \in L$ if and only if $(x', k') \in L'$,

(ii) $k' = g(k)$ is a function only of k, and

(iii) the computation is accomplished in time $f(k)n^{\alpha}$, where $n = |x|$, α is a constant independent of both n and k, and f is an arbitrary function.

Hardness for $W[1]$ is the working criterion that a parameterized problem is unlikely to be *FPT*. The k-CLIQUE problem is $W[1]$-complete, and often provides a convenient starting point for $W[1]$-hardness demonstrations.

The main degree sequence of parameterized complexity is

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

There are only the barest beginnings of a structure theory of parametric intractability. Anyone interested in this area should take the recent work of Flum and Grohe as a fundamental reference [FG01], as well as the few investigations exposited in [DF98].

3 Connections to Practical Computing and Heuristics

What is practical computing, anyway? A thought-provoking, instructive (and amusing) account of this neglected subject 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 "war stories" of practical computing, such as reported by Weihe, we are quickly forced to give up the idea that the real inputs (for most problems) fill up the definitional spaces of our mathematical modeling. The general rule also is that real inputs are *not random*, but rather have lots of hidden structure that may not have a familiar name or conceptualization, even if you knew what the hidden structure was. Weihe describes 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, so the straightforward application of the parameterized complexity program seems to fail as well.

However, the following two reduction rules can be applied to simplify (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.

(i) If $N(s) \subseteq N(s')$ then delete s.

(ii) 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 by brute force.

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

Weihe's example displays a universally applicable coping strategy for hard problems: *smart pre-processing*. It would be silly *not* to undertake preprocessing for an *NP*-hard problem, even if the next phase is simulated annealing, neural nets, roaming ants, genetic, memetic or the kitchen sink. In

a precise sense, this is *exactly* what fixed-parameter tractability is all about. The following is an equivalent definition of *FPT* [DFS99].

Definition 3.1 A parameterized language L is *kernelizable* if there is there is a parametric transformation of L to itself that satisfies:

(i) 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),

(ii) $k' \leq k$, and

(iii) $|x'| \leq h(k)$, where h is an arbitrary function.

Lemma 3.2 *A parameterized language L is fixed-parameter tractable if and only if it is kernelizable.*

Weihe's example looks like an *FPT* kernelization, but what is the parameter? As a thought experiment, let us define $K(G)$ for a bipartite graph G to be the maximum size of a component of G when G is reduced according to the two simple reduction rules above. Then it is clear, although it might seem artificial, that Hitting Set can be solved optimally in *FPT* time for the parameter $K(G)$. We can add this new tractable parameterization of Hitting Set to the already known fact that Hitting Set can be solved optimally in *FPT* for the parameter *treewidth*.

As an illustration of the power of pre-processing, the reader will easily discover a reduction rule for Vertex Cover that eliminates all vertices of degree 1. Not so easy is to show that all vertices of degree ≤ 3 can be eliminated, leaving as a kernel a graph of minimum degree 4. This pre-processing routine yields the best known heuristic algorithm for the general Vertex Cover problem (i.e., no assumption that k is small), and also plays a central role in the best known *FPT* algorithm for Vertex Cover.

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

It may seem that the definition of *FPT* allows too much pathology, since the parameter function $f(k)$ may be arbitrarily horrible. For example, there is an *FPT* algorithm for the Breakpoint Phylogeny problem, for the natural parameter k taken to be the total cost of the (Steiner) tree that comprises the solution. (The definition of the problem is not important to this discussion.) In practice this is frequently bounded by $k \leq 50$. The running time of this *FPT* algorithm is $f(k) \cdot n^2$ where $f(k) = (k!)^3$ or thereabouts. One might

suspect that this is not a very useful algorithm. But in fact, we are only reporting on an *ex post facto* analysis of an algorithm that has already been implemented, that is routinely in use, and that is rightly considered state-of-the-art [MWBWY00,CJMRWWW00]. Such examples do not seem to be uncommon. Many "heuristic" algorithms currently in use are turning out to be *FPT* algorithms for natural and relevant parameters. Those who design *FPT* algorithms should keep in mind that their $f(k)$'s are only the best they are able to prove concerning a worst-case analysis, and that their algorithms may in fact be much more useful in practice than the pessimistic analysis indicates, on realistic inputs, particularly if any nontrivial kernelization is involved.

4 Some Research Frontiers

In this section we discuss some current research directions in parameterized complexity.

4.1 The Complexity of Approximation

The emphasis in the vast area of research on polynomial-time approximation algorithms is concentrated on the notions of:

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

The connections between the *parameterized complexity* and *polynomial-time approximation* programs are 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 $O(n^{35/\epsilon^3})$. Thus for a 20% error we are looking at a "polynomial-time" algorithm with a running time of $O(n^{4,275})$. The parameter $k = 1/\epsilon$ is one of the most important and obvious in all of the theory of computing.

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 4.1 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 $k = 1/\epsilon$.

The following important theorem was first proved by Cristina Bazgan in her Master's Thesis (independently by Cesati and Trevisan) [Baz95,CT97].

Theorem 4.2 *Suppose that Π_{opt} is an optimization problem, and that Π_{param} is the corresponding parameterized problem, where the parameter is the value*

of an optimal solution. Then Π_{param} *is fixed-parameter tractable if* Π_{opt} *has an efficient PTAS.*

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

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

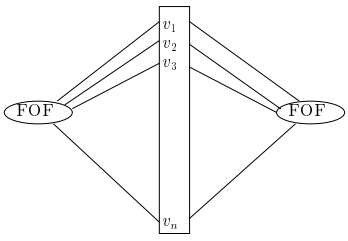
The following theorem is recent joint work with Cai, Juedes and Rosamond.

Theorem 4.3 *Planar TMIN does not have an EPTAS unless* $FPT = W[1]$.

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

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

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

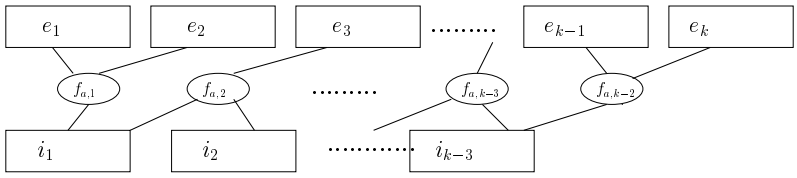


We describe the construction in two stages. In the first stage, we use k blocks of n variables and a collection C' of $k(k-1)/2+k$ FOFs. In a weight k satisfying assignment for C', exactly one variable v_i, j in each block of variables $b_i = [v_{i,1}, \ldots, v_{i,n}]$ will be set to true. We interpret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

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

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

The widget A_k can be drawn as follows.

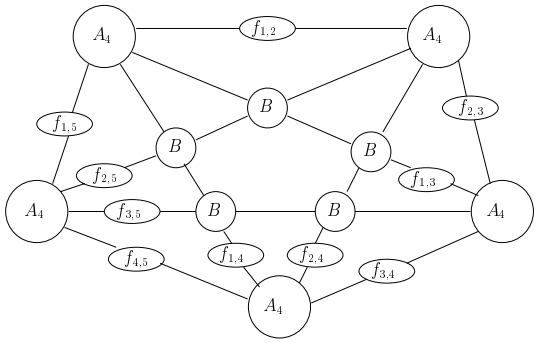


Since each internal block is connected to exactly two FOFs, the incidence graph for this widget can be drawn on the plane without crossing any edges.

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

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

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



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

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

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

proof.

□

In a similar manner the other two planar logic problems defined by Khanna and Motwani can be shown to be $W[1]$-hard. PTAS's for these problems are therefore likely never to be very useful, since the goodness of the approximation must apparently be paid for in the exponent of the polynomial running time.

A Second Connection. There is a second strong connection between parameterized complexity and polynomial-time approximation because 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 export bridge from the former to the latter was first pointed out in [NSS98]. See also [FMcRS01].

4.2 The Art and Science of Kernelization in the Design of Heuristics and Practical Algorithms

The toolkit for proving *FPT* includes a number of deep and distinctive methods: well-quasi-ordering, color-coding, bounded treewidth — as well as the elementary methods of kernelization and search trees. (All of these are exposited in [DF98].) Mathematically, these positive methods are extremely interesting and powerful at classifying problems as fixed-parameter tractable. For the purposes of practical algorithm design, *reduction to a problem kernel* is probably the single most important contribution to the systematic design of heuristics. This is in some sense a comprehensive connection, since *fixed-parameter tractability* is equivalent to *kernelizability* as shown by Lemma 1 of §3.

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

(1) Kernelization rules are frequently surprising in character, laborious to prove, and nontrivial to discover. Once found, they are small gems that remain permanently in the heuristic design file for hard problems. No one concerned with an application of Hitting Set should henceforth neglect Weihe's preprocessing rules, for example. The kernelization for Vertex Cover to graphs of minimum degree 4, for another example, includes the following nontrivial transformation. Suppose G has a vertex x of degree 3 that has three mutually nonadjacent neighbors a, b, c. Then G can be simplified by: (1) deleting x, (2) adding edges from c to all the vertices in $N(a)$, (3) adding edges from a to all the vertices in $N(b)$, (3) adding edges from b to all the vertices in $N(c)$, and (4) adding the edges ab and bc. Note that this transformation is not even symmetric! The resulting graph G' has a vertex cover of size k (which need not be assumed to be small) if and only if G has a vertex cover of size k.

Moreover, an optimal or good approximate solution for G' lifts constructively to an optimal or good approximate solution for G. The research direction this points to is **to discover these gems of smart preprocessing for all of the hard problems**. There is absolutely nothing to be lost in smart preprocessing, no matter what the subsequent phases of the algorithm (even if the next phase is genetic algorithms or simulated annealing).

(2) Kernelization rules cascade in ways that are surprising, unpredictable in advance, and often quite powerful. Finding a rich set of reduction rules for a hard problem may allow the synergistic cascading of the pre-processing rules to "wrap around" hidden structural aspects of real input distributions. Weihe's train problem provides an excellent example. According to the experience of Alber, Gramm and Niedermeier with implementations of kernelization-based *FPT* algorithms [AGN01], the effort to kernelize is amply rewarded by the subsequently exponentially smaller search tree. Similar results have also been reported by Moret *et al.* with respect to the Breakpoint Phylogeny problem [MWBWY00].

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

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

References

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

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

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

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

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

[AFMRRRS01] 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 2001.

[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.

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

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

[CJ01] Liming Cai and D. Juedes, "Subexponential Parameterized Algorithms Collapse the W-Hierarchy," in: Proceedings of ICALP 2001, Crete, Greece, LNCS 2076 (2001).

[CJMRWWW00] M. Cosner, R. Jansen, B. Moret, L. Raubeson, L. Wang, T. Warnow and S. Wyman, "A New Fast Heuristic for Computing the Breakpoint Phylogeny and Experimental Phylogenetic Analyses of Real and Synthetic Data," manuscript, 2000.

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

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

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

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

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

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

[DF99] R. Downey and M. Fellows, "Parameterized Complexity After Almost Ten Years: Review and Open Questions," in: *Combinatorics, Computation and Logic, DMTCS'99 and CATS'99*, Australian Computer Science Communications, Springer-Verlag Singapore, vol. 21 (1999), 1–33.

[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, "Solving Large FPT Problems on Coarse Grained Parallel Machines," manuscript, 2001.

[Fe01] H. Fernau, "Remarks on Parameterized Enumeration," manuscript, 2001.

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

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

[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.

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

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

[GSS01] G. Gottlob, F. Scarcello and M. Sideri, "Fixed Parameter Complexity in AI and Nonmonotonic Reasoning," to appear in *The Artificial Intelligence*

Journal. Conference version in: *Proc. of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning* (LPNMR'99), vol. 1730 of *Lecture Notes in Artificial Intelligence* (1999), 1–18.

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

[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.

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

[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).

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

[MWBWY00] B. Moret, S. Wyman, D. Bader, T. Warnow and M. Yan, "A New Implementation and Detailed Study of Breakpoint Analysis," manuscript, 2000.

[Nie98]
R. Niedermeier, "Some Prospects for Efficient Fixed-Parameter Algorithms," *Proc. SOFSEM'98* Springer-Verlag LNCS 1521 (1998), 168–185.

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

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

[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*.

[Ra97] V. Raman, "Parameterized Complexity," in: *Proceedings of the 7th National Seminar on Theoretical Computer Science*, Chennai, India (1997), 1–18.

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