Discrete Mathematics, Vol. 229(1–3), pp. 3–27, 2001

Faster Exact Algorithms for Hard Problems: A Parameterized Point of View [1]

Jochen Alber Jens Gramm Rolf Niedermeier [2]

Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, Sand 13, D-72076 Tübingen, Fed. Rep. of Germany

Abstract

Recent times have seen quite some progress in the development of "efficient" exponential time algorithms for *NP*-hard problems. These results are also tightly related to the so-called theory of fixed parameter tractability. In this incomplete, personally biased survey, we reflect on some recent developments and prospects in the field of fixed parameter algorithms.

Key words: *NP*-complete problems, parameterized complexity, fixed parameter tractability, *W*-hierarchy, exact algorithms

1 Introduction

How to cope with computational intractability? Several methods to deal with this problem have been developed: approximation algorithms [1,60], average case analysis [62], randomized algorithms [70], and heuristic methods [69]. All of them have their drawbacks, such as the difficulty of approximation, lack of mathematical tools and results, limited power of the method itself, or the lack of provable performance guarantees at all. Clearly, the direct way of attacking

[1] Parts of this work are an updated and revised version of the invited paper "Some prospects for efficient fixed parameter algorithms" of the third author in the proceedings of the 25th Conference on Current Trends in Theory and Practice of Informatics (SOFSEM'98), Springer, *LNCS* 1521, pages 168–185, held in Jasna, Slovakia, November 21–27, 1998. This work is performed within the "PEAL" project (Parameterized complexity and Exact ALgorithms), supported by the Deutsche Forschungsgemeinschaft (NI-369/1-1).
[2] Corresponding author.
E-mail address: niedermr@informatik.uni-tuebingen.de

NP-hard problems is in providing deterministic, exact algorithms. However, in this case, one has to deal with exponential running times. Currently, there is an increasing interest in faster exact solutions for *NP*-hard problems. In particular, performance bounds are to be *proven.* Despite their exponential running times, these algorithms may be interesting from a theoretical as well as a practical point of view. With respect to the latter, note that for some applications, really exact solutions are needed or the input instances are of modest size, so that exponential running times can be tolerated.

Parameterized complexity theory, whose cantus firmus can be characterized by the words "not all forms of intractability are created equal" [34], is another proposal on how to cope with computational intractability in some cases. In a sense, so-called "fixed parameter algorithms" form a variant of exact, exponential time solutions mainly for *NP*-hard problems. This is the basic subject of this paper and some related survey articles [35,36,83].

Many hard computational problems have the following general form: given an object x and a positive integer k, does x have some property that depends on k? For instance, the *NP*-complete VERTEX COVER problem is: given an undirected graph $G = (V, E)$ and a positive integer k, does G have a vertex cover of size at most k? Herein, a vertex cover is a subset of vertices $C \subseteq V$ such that each edge in E has at least one of its endpoints in C. In parameterized complexity theory, k is called the *parameter*. In many applications, the parameter k can be considered to be "small" in comparison with the size $|x|$ of the given object x. Hence, it may be of high interest to ask whether these problems have deterministic algorithms that are exponential *only* with respect to k and polynomial with respect to $|x|$.

The basic observation of parameterized complexity, as chiefly developed by Downey and Fellows [34], is that for many hard problems, the seemingly inherent "combinatorial explosion" really can be restricted to a "small part" of the input, the parameter. So, for instance, VERTEX COVER allows for an algorithm with running time $O(kn + 1.271^k)$ [24,75], where the parameter k is a bound on the maximum size of the vertex cover set we are looking for and n is the number of vertices of the given graph. The best known "non-parameterized" solution for VERTEX COVER is due to Robson [84]. He showed that INDEPENDENT SET and, thus, VERTEX COVER can be solved in time $O(1.211^n)$. However, for $k \leq 0.79\,n$, the above mentioned fixed parameter solution turns out to be better. Moreover, note that in several applications $k \ll n$ is a natural assumption, underpinning the qualities of fixed parameter solutions further.

In what follows, we assume the reader to be familiar with basic notions from algorithms and complexity as, e.g., provided by the text books [28,48,68,79]. The aim of this work is not to list as many parameterized problems as possible together with their "fixed parameter solutions," but to give a, in a sense, "personally biased" view on current research including open problems.

The paper is structured as follows. In the next section, we very briefly provide a general overview of some main topics and ideas of parameterized complexity theory. Moreover, we take a closer look at the concept of "fixed parameter tractability" and also the critical points in it. Thus, we will have the ground for the rest of the paper. In Sections 3–6, we sketch some results and challenges in the fields of graph theory, VLSI design, computational biology, and logic, thus showing the richness of the field for applying fixed parameter techniques. The idea here is to present some fundamental results, ideas, and observations by examples, without even trying to give a complete account of the whole field. Each of these sections contains two subsections, the first of which providing some known results and the second one presenting some possible challenges for future research. Finally, in Section 7 we discuss the so far relatively neglected field of implementing fixed parameter algorithms. We end the paper by drawing some general conclusions and prospects for future research.

2 A crash course in parameterized complexity

We briefly want to sketch some complexity theoretical aspects for parameterized problems. For a detailed and complete exposition we refer to the monograph of Downey and Fellows [34]. The focus of this section, however, lies on the practical relevance of fixed parameter tractability.

2.1 Some theory

Given an undirected graph $G = (V, E)$ with vertex set V, edge set E and a positive integer k, the NP-complete VERTEX COVER problem is to determine whether there is a subset of vertices $C \subseteq V$ with k or fewer vertices such that each edge in E has at least one of its endpoints in C. VERTEX COVER is *fixed parameter tractable*: There are algorithms solving it in time less than $O(kn + 1.3^k)$ [24,75,77]. By way of contrast, consider the also NP-complete CLIQUE problem: Given an undirected graph $G = (V, E)$ and a positive integer k, CLIQUE asks whether there is a subset of vertices $C \subseteq V$ with k or fewer vertices such that C forms a clique by having all possible edges between the vertices in C. CLIQUE appears to be *fixed parameter intractable*: It is *not* known whether it can be solved in time $f(k)n^{O(1)}$, where f might be an arbitrarily fast growing function only depending on k [34]. Thus, in some sense, the handling of the parameter seems to be more sophisticated for the CLIQUE problem compared to the VERTEX COVER problem. Moreover, unless $P = NP$, the well-founded conjecture is that no such algorithm exists. The best known algorithm solving CLIQUE runs in time $O(n^{ck/3})$ [73], where c is the exponent on the time bound for multiplying two integer $n \times n$ matrices (currently best known, $c = 2.38$, see [27]). Note that $O(n^k)$ time is trivial. The decisive point is that k appears in the exponent of n, and there seems to be no way "to shift the combinatorial explosion only into k," independent from n [34–36,83].

The observation that NP-complete problems like VERTEX COVER and CLIQUE behave completely differently in a "parameterized sense" lies at the very heart of parameterized complexity, a theory pioneered by Downey and Fellows and some of their co-authors [34]. In this paper, we will mainly concentrate on the world of fixed parameter tractable problems as, e.g., exhibited by VERTEX COVER. Hence, here we only briefly sketch some very basics from the theory of parameterized intractability in order to provide some background on parameterized complexity theory and the ideas behind it. For many further details and an additional discussion, we refer to the literature, e.g., [34–36,83].

Attempts to prove nontrivial, absolute lower bounds on the computational complexity of problems have made relatively little progress [16]. Hence, it is not surprising that up to now there is no proof that *no* $f(k)n^{O(1)}$ time algorithm for CLIQUE exists. In a more complexity theoretical language, consider the class of parameterized problems that can be solved in deterministic time $f(k)n^{O(1)}$, called FPT. This can be rephrased by saying that it is unknown whether CLIQUE $\in FPT$. The complexity class FPT is called the set of *fixed parameter tractable* problems. Analogously to classical complexity theory, Downey and Fellows developed some way out of this quandary by providing a completeness program. However, the completeness theory of parameterized intractability involves significantly more technical effort. We very briefly sketch some integral parts of this theory in the following.

To start with a completeness theory, we first need a reducibility concept: Let $L, L' \subseteq \Sigma^* \times \mathbf{N}$ be two parameterized languages. [3] For example, in the case of CLIQUE, the first component is the input graph coded over some alphabet Σ and the second component is the positive integer k, that is, the parameter. We say that *L reduces to L' by a standard parameterized m-reduction* if there are functions $k \mapsto k'$ and $k \mapsto k''$ from $\mathbf{N}$ to $\mathbf{N}$ and a function $(x, k) \mapsto x'$ from $\Sigma^* \times \mathbf{N}$ to Σ^* such that

(1) $(x, k) \mapsto x'$ is computable in time $k''|x|^c$ for some constant c and
(2) $(x, k) \in L$ iff $(x', k') \in L'$.

[3] In most of the cases it is natural to have positive integers as a parameter. However, in some cases different structures appear as parameter (e.g., a graph as in the GRAPH MINOR ORDER TESTING problem [34]). Here, we define a parameterized language to be a language in $\Sigma^* \times \Sigma^*$.

Notably, most reductions from classical complexity turn out *not* to be parameterized [34]. For instance, the well-known reduction from INDEPENDENT SET to VERTEX COVER, which is given by letting $G' = G$ and $k' = |V| - k$ for a graph instance $G = (V, E)$ (see [79]), is not a parameterized one. This is due to the fact that the reduction function of the parameter $k \mapsto k'$ strongly depends on the instance G itself, hence contradicting the definition of a parameterized m-reduction. However, the reductions from INDEPENDENT SET to CLIQUE and the other way round, which are obtained by simply passing the original graph over to the complementary one for $k' = k$, actually turn out to be parameterized ones. Therefore, these problems are of comparable difficulty in terms of parameterized complexity.

Now, the "lowest class of parameterized intractability", so-called $W[1]$, can be defined as the class of languages that reduce to the so-called SHORT TURING MACHINE ACCEPTANCE problem (also known as the k-STEP HALTING problem) [34]. Here, we want to determine for an input consisting of a nondeterministic Turing machine M and a string x, whether M has a computation path accepting x in at most k steps. In some sense, this is the parameterized analogue to the TURING MACHINE ACCEPTANCE problem—the basic generic *NP*-complete problem in classical complexity theory. Some more naturally defined problems being $W[1]$-*complete* are the CLIQUE and the INDEPENDENT SET problem. Note that a parameterized m-reduction from one of these problems to VERTEX COVER would lead to a collapse of the classes $W[1]$ and FPT. Downey and Fellows provide an extensive list of many more $W[1]$-complete problems [34].

As a matter of fact, a whole hierarchy of parameterized intractability can be defined, $W[1]$ only being the lowest level. In general, the classes $W[t]$ are defined based on "logical depth" (i.e., the number of alternations between unbounded fan-in And- and Or-gates) in boolean circuits. For instance, the well-known DOMINATING SET problem, which is *NP*-complete, is known to be $W[2]$-complete when taking the size of the dominating set as a parameter [34]. We omit any further details in this direction and just refer to the monograph of Downey and Fellows [34].

We also want to mention that Flum and Grohe recently came up with some close connections between parameterized complexity theory and a general logical framework of descriptive complexity theory [45,54]. They study the parameterized complexity of various model-checking problems through the syntactical structure of the defining sentences. Moreover, they provide a descriptive characterization of the class FPT, as well as an approach of how to characterize classes of intractability by syntactical means.

There exists a very rich structural theory of parameterized complexity, somewhat similar to classical complexity. Observe, however, that in some respects parameterized complexity appears to be, in a sense, "orthogonal" to classical complexity: For instance, the so-called problem of computing the V-C dimension from learning theory [13,80], which is not known (and not believed) to be *NP*-hard, is $W[1]$-complete [32,33]. Thus, although in the classical sense it appears to be easier than VERTEX COVER (which is *NP*-complete), the opposite appears to be true in the parameterized sense, because VERTEX COVER is in *FPT*.

From a practical point of view, it is probably sufficient to distinguish between $W[1]$-hardness and membership in *FPT*. Thus, for an algorithm designer not being able to show fixed-parameter tractability of a problem, it may be sufficient to give a reduction from, e.g., CLIQUE to the given problem using a standard parameterized m-reduction. This then gives a concrete indication that, unless $P = NP$, the problem is unlikely to allow for an $f(k)n^{O(1)}$ time algorithm. One piece of circumstantial evidence for this is the result showing that the equality of $W[1]$ and *FPT* would imply a time $2^{o(n)}$ algorithm for the *NP*-complete 3-CNF-SAT problem [34], which would mean a breakthrough in computational complexity theory.

2.2 Interpreting fixed parameter tractability

The remainder of this paper concentrates on the world inside *FPT* and the potential it carries for improvements and future research. We therefore finish this section by an interpretation of the class *FPT* under some application aspects. Note that in the definition of *FPT* the function $f(k)$ may take unreasonably large values, e.g.,

$$2^{2^{2^{2^{2^{2^{2^{k}}}}}}}\,.$$

Thus, showing that a problem is a member of the class *FPT* does not necessarily bring along an efficient algorithm (not even for small k). In fact, many problems that are classified fixed parameter tractable still wait for such efficient, hence practical algorithms. In this sense, we strongly have to distinguish two different aspects of fixed parameter tractability: The theoretical part which consists in classifying parameterized problems along the W-hierarchy (i.e., proving membership in *FPT*) and the algorithmic component of actually finding efficient algorithms for problems inside the class *FPT*.

The Graph Minor Theorem by Robertson and Seymour [34, chapter 7], for example, provides a great tool for the classification of graph problems. It states that, for a given family of graphs $\mathcal{F}$ which is closed under taking minors, membership of a graph G in $\mathcal{F}$ can be checked by analyzing whether a certain finite "obstruction set" appears as a minor in G. Moreover, the GRAPH MINOR ORDER TESTING problem is in *FPT* [34], or, more precisely, for a fixed graph

Table 1
Comparing the efficiency of various VERTEX COVER algorithms with respect to the exponential terms involved.

k	$f(k) = 2^k$	$f(k) = 1.292^k$	$f(k) = 1.271^k$
10	$\approx 10^3$	≈ 13	≈ 11
20	$\approx 10^6$	≈ 170	≈ 120
30	$\approx 10^9$	≈ 2180	≈ 1330
40	$\approx 10^{12}$	$\approx 2.8 \cdot 10^4$	$\approx 1.5 \cdot 10^4$
50	$\approx 10^{15}$	$\approx 3.7 \cdot 10^5$	$\approx 1.6 \cdot 10^5$
75	$\approx 10^{22}$	$\approx 2.2 \cdot 10^8$	$\approx 6.5 \cdot 10^7$
100	$\approx 10^{30}$	$\approx 1.3 \cdot 10^{11}$	$\approx 2.6 \cdot 10^{10}$
500	$\approx 10^{150}$	$\approx 4.2 \cdot 10^{55}$	$\approx 1.2 \cdot 10^{52}$

G of size n, there is an algorithm with running time $O(f(|H|)n^3)$ that decides whether a graph H is a minor or not. As a matter of fact, the set $\mathcal{F}_k$ of vertex covers of size $\leq k$ are closed under taking minors, hence there exists a finite obstruction set $\mathcal{O}_k$ for $\mathcal{F}_k$. The above method then yields the existence of a fixed parameter algorithm for VERTEX COVER. However, in general, the function f appearing in the GRAPH MINOR ORDER TESTING algorithm grows fast and the constants hidden in the O-notation are huge. Moreover, finding the obstruction set in the Robertson-Seymour Theorem, in general, is highly non-constructive. Thus, the above mentioned method may only serve as a classification tool.

Taking into account that the theory of fixed parameter tractability should be understood as an approach to cope with inherently hard problems, it is necessary to aim for practical, efficient fixed parameter algorithms. In the case of VERTEX COVER, for example, it's fairly easy to come up with an algorithm of running time $O(2^k n)$ using a simple search tree strategy. The base of the exponential term in k was further improved, now below 1.3 (see [24,77,89]). Table 1 relates the size of the exponential term for these base values. From this table we can conclude that even seemingly minor improvements in the exponential growth of f can lead to significant changes in the running time and, moreover, enlarge the range of suitable values k for which the given algorithm guarantees small running time.

In order to address this issue of efficient functions f in a more detailed manner, Downey and Fellows [34] introduced so-called klam values. The *klam* value of a fixed parameter algorithm is the largest value of k such that the function f occurring in the algorithm's running time fulfills $f(k) \leq U$, where U is some reasonable absolute bound on the maximum number of steps of any computation, e.g., $U = 10^{20}$. For example, using $U = 10^{20}$, the current klam value for VERTEX COVER is approximately 140. Unfortunately, for few parameterized problems klam values of comparable high quality are known. Hence, an important algorithmic challenge concerning FPT problems is to provide klam values as large as possible.

In this context, to demonstrate the problematic nature of the comparison "fixed parameter tractable" versus "fixed parameter intractable," let us compare the functions 2^{2^k} and $n^k = 2^{(k \log n)}$. The first refers to fixed parameter tractability, the second to intractability. It is easy to verify that assuming input sizes n in the range from 10^3 up to 10^{15}, the value of k where 2^{2^k} starts to exceed n^k is in the small range $\{6, 7, 8, 9\}$. A striking example in this direction is that of computing treewidth. For constant k, there is a famous result giving a linear time algorithm to compute whether a graph has treewidth at most k [14]. However, this algorithm suffers from enormous constant factors (unless $k \leq 3$) and so the $O(n^{k+1})$ algorithm [6] is more practical. Hence, this shows how careful one has to be with the term fixed parameter tractable, since, in practice with reasonable input sizes, a fixed parameter intractable problem can easily turn out to have a still more "efficient" solution than a fixed parameter tractable one.

3 Graph theoretical problems

Graph theory and related computational problems so far appear to be among the most fertile grounds for the study of parameterized problems.

3.1 Some results

Vertex cover: Given an undirected graph $G = (V, E)$ with n vertices and a positive integer k, is there a subset $C \subseteq V$ of vertices such that each edge in E has at least one endpoint in C? VERTEX COVER has seen quite a long history in the development of FPT algorithms [34, page 5]. Surprisingly, a lot of papers published fixed parameter results on VERTEX COVER that are worse than the $O(2^k n)$ time bound that directly follows from the elementary search tree method e.g. described in Mehlhorn's text book on graph algorithms [68, page 216]. This simple observation, which is also used in the factor 2 approximation algorithm for VERTEX COVER, is as follows: Each edge $\{a, b\}$ has to be covered. Hence, either a or b (or both) has to be in the cover. Thus, building a search tree where we branch as either bringing a or b in the cover, deleting the respective vertex together with its incident edges and continuing recursively with the remaining graph by choosing the next edge (which is arbitrary) solves the problem. The search tree, which has depth k, has 2^k nodes, each of which can be processed in linear time using suitable data structures. In recent times, there has been very active research on lowering the size of the search tree [9,24,36,77,89], the best known result now being 1.271^k [24]. The basic idea behind all of these papers is to use two fundamental techniques of parameterized complexity, that is, *bounded search trees* and *reduction to problem kernel.* To improve the search tree size, intricate case distinctions with respect to the degree of graph vertices were designed. As to reduction to problem kernel, the idea is as follows: Assume that you have a graph vertex v of degree $k+1$, that is, $k+1$ edges have endpoint v. Then to cover all these edges, one has to either bring v or all its neighbors into the vertex cover. In the latter case, however, the vertex cover then would have size at least $k+1$—too much if we are only interested in covers of size at most k. Hence, without branching we *have* to bring v and all other vertices of degree greater than k into the cover. From this observation, one can easily conclude that after doing this kind of preprocessing, the remaining graph may consist of $O(k^2)$ vertices. A well-known theorem of Nemhauser and Trotter [72,82] can be used to construct an improved reduction to problem kernel resulting in a graph of only $2k$ vertices [24].

Up to now, research concentrated on UNWEIGHTED VERTEX COVER. However, very recently, first results were also obtained for three variants of WEIGHTED VERTEX COVER [78]. Here, by way of contrast to UNWEIGHTED VERTEX COVER, each vertex has some positive number as weight and we are looking for a cover such that the sum of the weights of the vertices in the cover is at most k. The three variants are:

(1) INTEGER-WVC, where the weights are arbitrary positive integers,
(2) REAL-WVC, where the weights are real numbers ≥ 1, and
(3) GENERAL-WVC, where the weights are positive real numbers.

GENERAL-WVC easily turns out to be fixed parameter intractable unless $P = NP$, whereas REAL-WVC can be solved in time $O(kn + 1.3954^k)$ and INTEGER-WVC as fast as UNWEIGHTED VERTEX COVER [78]. To derive the bounds for REAL-WVC is clearly the most important result. In addition, if we modify the REAL-WVC problem such that k is *not the weight* of the vertex cover we are looking for, but the *number of vertices* in a minimum weight vertex cover, then also the running time $O(kn + 1.3954^k)$ can be obtained.

Planar dominating set: The DOMINATING SET problem is, given a graph $G = (V, E)$ and a positive integer k, is there a set of k vertices $V' \subseteq V$ with the property that every vertex of G either belongs to V' or has a neighbor in V'? DOMINATING SET is $W[2]$-complete [34] and, thus, considered to be fixed parameter intractable. For planar graphs, however, Downey and Fellows [33,34] gave an $O(11^k n)$ bounded search tree algorithm. Note that, in contrast to VERTEX COVER, for PLANAR DOMINATING SET it is not obvious how to construct such a bounded search tree. The approach by Downey and Fellows is based on a clever graph theoretical observation (see [34, Lemma 3.3]). Unfortunately, even for a modest k, the exponential term 11^k gets enormously big. Hence, as in the VERTEX COVER case, one would be very interested in lowering the exponential term.

A completely different approach, which does not rely on a bounded search tree strategy, was presented recently [3]. There, it is shown that PLANAR DOMINATING SET can be solved in time $3^{O(\sqrt{k})}n$. Besides, fixed parameter algorithms of similar running times can be obtained for a whole series of related problems on planar graphs, such as the DOMINATING SET WITH PROPERTY P (which includes INDEPENDENT DOMINATING SET, TOTAL DOMINATING SET, PERFECT CODE), or the so-called FACE COVER problem (for details we refer to [3] and the references therein).

The key in order to obtain these results is a—to some extent unexpected—theorem on the relation of the domination number $\gamma(G)$ (which is the minimum size of a dominating set admitted by G) and the treewidth $tw(G)$ of a planar graph: It can be proven that $tw(G) = O(\sqrt{\gamma(G)})$. Moreover, this not being a pure existence theorem, one can show that a corresponding tree decomposition can be constructed in time $O(\sqrt{\gamma(G)}n)$. The method is based on "small separator" techniques (see [3] for details).

The overall algorithm then consists of two stages: the first stage finds the tree decomposition of bounded width and the second stage solves the problem using well-known dynamic programming approaches on the tree decomposition (see, e.g., [15,90,91]). This two-stage strategy serves as a common tool to design fixed parameter algorithms for a variety of different graph theoretical problems. However, in general, the first stage of this type of algorithms, i.e., the problem to determine whether a graph has treewidth bounded by some constant ℓ and, if so, producing a corresponding tree decomposition, has, at the current state of research, no algorithmically feasible solution. Even though the problem is proven to be fixed parameter tractable, the constants in the algorithm presented in [14] are too large for practical purposes. In this sense, the PLANAR DOMINATING SET plays an exceptional role, where a tree decomposition of small width, namely $O(\sqrt{\gamma(G)})$, can be found quickly, namely in $O(\sqrt{\gamma(G)}n)$ time.

The resulting algorithm presented in [3] has time complexity $3^{O(\sqrt{k})}n$. Although the constant hidden in the O-notation turns out to be $6\sqrt{34}$, and thus, at first glance, the practical usefulness of this algorithm is uncertain, the point is that we have a significant asymptotic improvement. From a theoretical point of view, this means more than "only" lowering the constant base of the exponential term. To our knowledge, this is the first non-trivial result for an

NP-hard problem which admits a fixed parameter algorithm with a sublinear term in the exponent of the running time. It is not known, whether there is a similar result for the general VERTEX COVER problem (note that in the case of PLANAR VERTEX COVER again a sublinear expression in the exponential term can be obtained using the techniques presented in [3]).

Minimum fill-in: The *NP*-complete MINIMUM FILL-IN problem asks whether a graph can be triangulated by adding at most k edges. Kaplan *et al.* [64] developed a search-tree based $O(2^{4k}m)$ time algorithm (which improves to $O((4^k/(k+1)^{3/2})(m+n))$ due to a refined analysis by Cai [22]) and a more intricate $O(k^2nm + k^6 2^{4k})$ algorithm for the problem. Here, n denotes the number of vertices and m denotes the number of edges in the graph. This also illustrates that it can be very important (and difficult!) to make the exponential term "additive" (as in the second case) instead of only "multiplicative" (as in the first case). In addition, Kaplan *et al.* show that PROPER INTERVAL GRAPH COMPLETION (motivated by computational biology) and STRONGLY CHORDAL GRAPH COMPLETION, both *NP*-hard, are fixed parameter tractable (see [64] for details).

Longest path or cycle: Finally, let us mention in passing another promising method yielding fast fixed parameter algorithms for certain graph problems: the so-called technique of color coding. Using this technique, Alon *et al.* [4] came up with an $2^{O(k)}|E|\log|V|$ time algorithm for the LONGEST PATH problem in a graph $G = (V, E)$. Here, we ask for a simple path of length $k-1$ in G, where "simple" means that each vertex appears at most once on this path. The approach by Alon *et al.* is based on the idea of hashing and can be sketched as follows: Fix an arbitrary coloring (that is, a hash function) of G using k colors. Then, making use of an algorithm based on dynamic programming, it's relatively simple to obtain an $O(2^k|E|)$ time algorithm that finds a path of length k, such that all vertices along the path have distinct colors. Such a path is called colorful. Now, each colorful path clearly is simple. Conversely, in order to come up with a simple path, the above algorithm has to be repeated using sufficiently many different colorings. Thus, the hard part of this approach consists in showing that the size of the family of colorings that are necessarily needed can be kept small. The mainly combinatorial arguments for the construction of such a (so-called) k-perfect family of hash functions can be found in Schmidt and Siegel [85]. Note that the related problem of finding simple cycles of length k can be attacked with similar methods.

3.2 Some challenges

(1) Besides still asking for improvements of general vertex cover, can, e.g., VERTEX COVER for planar graphs be solved much faster? Possible ap-

proaches might arise from the general techniques presented in [3], combined with results of Baker [8] and the known algorithms for the general VERTEX COVER problem.

(2) How "practical" can the solutions for PLANAR DOMINATING SET be made? This question addresses two issues: On the one hand, it is still unclear how to obtain an $O(c^{\sqrt{k}}n)$ time algorithm with some reasonable constant c. On the other hand, it might be of high value to lower the base of the exponential term in the $O(11^k n)$ search tree algorithm presented by Downey and Fellows. Also, does PLANAR DOMINATING SET admit a polynomial size problem kernel which could be used to apply the "interleaving techniques" in [75]?

(3) The exponential term 4^k for MINIMUM FILL-IN still should be improved, and the same with the other problems mentioned in [64].

(4) How general and practical is the technique of COLOR CODING? Can it also be applied to other problems to show their fixed parameter tractability?

(5) In the t-VERTEX COVER problem one seeks to cover t instead of all edges [20,61]. What is the parameterized complexity of t-VERTEX COVER?

4 VLSI design problems

VLSI design is one more attractive field for parameterized complexity studies [34,42]. Interestingly, reduction to problem kernel, a basic parameterized technique, was earlier used in algorithms for VLSI design [38]. Due to our still limited insight and knowledge, we only mention very few aspects of fixed parameter algorithms for VLSI design.

4.1 Some results

Spare allocation for reconfigurable VLSI: Put concisely, this "most widely used approach to reconfigurable VLSI" uses spare rows and columns to tolerate failures in rectangular arrays of identical computational elements, which may be as simple as memory cells or as complex as processor units [65]. If a faulty cell is detected, the entire row or column is replaced by a spare one. The underlying problem can be easily modeled in a graph theoretical setting: given a bipartite graph $G = (V_1, V_2, E)$ and *two* positive integers k_1 and k_2, are there two subsets $C_1 \subseteq V_1$ and $C_2 \subseteq V_2$ of sizes $|C_1| \leq k_1$ and $|C_2| \leq k_2$ such that each edge in E has at least one endpoint in $C_1 \cup C_2$? The existence of *two* parameters and two vertex sets makes this problem, called CONSTRAINT BIPARTITE VERTEX COVER (CBVC), quite different from the original VERTEX COVER problem. Thus, whereas the classical VERTEX COVER (with only one parameter!) restricted to bipartite graphs is solvable in polynomial time (because there is a close relation to the polynomial time solvable maximum matching problem for bipartite graphs), by a reduction from CLIQUE it has been shown that CBVC is *NP*-complete [65]. Recently, to our knowledge, the first nontrivial fixed parameter algorithm for CONSTRAINT BIPARTITE VERTEX COVER was given, running in time $O(1.3999^{k_1+k_2} + (k_1 + k_2)n)$ [44]. To achieve the result, well-known methods from parameterized complexity were combined, namely reduction to problem kernel and bounded search trees, with a new technique for restricting the size of the search tree by deferring some work to a third, polynomial-time phase, but nevertheless counting the parameter reduction as kind of bonus when processing the search tree and hence reducing its size.

Cutwidth: This is another problem originating in VLSI design [34,42]. A *layout* of a graph $G = (V, E)$ is a one-to-one function $f : V \to \{1, \ldots, |V|\}$. If we regard $[1, |V|]$ as an interval on real numbers and consider $\alpha \in [1, |V|]$, then we call the number of edges $\{u, v\} \in E$ with $f(u) < \alpha$ and $f(v) > \alpha$ the value of the cut at α. The cutwidth of a layout f of G then is the maximum of the value of the cut over all α. The CUTWIDTH problem for G is to find the minimum of the cutwidths of all possible layouts of G. The decision version of this problem is *NP*-complete [48]. In the parameterized version, for a given positive integer k we ask whether a given graph has cutwidth $\leq k$. It is known that CUTWIDTH is fixed parameter tractable [34], but the bounds on the exponential term seem to be huge.

Bandwidth: The classical problem from VLSI design, BANDWIDTH, is $W[t]$-hard for all t [34] and thus appears to be fixed parameter intractable. The bandwidth of a layout f of G is the maximum of $|f(u) - f(v)|$ over all edges $\{u, v\} \in E$. The bandwidth of G then is the minimum bandwidth of all possible layouts of G. The decision version of this problem is *NP*-complete [48]. Again, the parameterized version asks, given a graph $G = (V, E)$ and a positive integer, does G have bandwidth $\leq k$? Despite of its great practical importance, bandwidth seems to be a problem where parameterized complexity studies cannot help in general. A recent survey paper [40] sketches various approaches how to cope with the hardness of bandwidth, rising many open questions also concerning the development of exact algorithms.

4.2 Some challenges

(1) There is a variant of CONSTRAINT BIPARTITE VERTEX COVER with three instead of two parameters, motivated by reconfigurable programmable logic arrays [58]. This problem deserves parameterized complexity investigations similar to that undertaken for CBVC.
(2) The practical validation of the proposed algorithm for CONSTRAINT BI-

partite vertex cover, which is based on a complicated case distinction, remains to be done. In particular, average case analyses, also based on special input instances (cf. [87]), are desirable.

(3) Give concrete, small bounds for the exponential term in a fixed parameter algorithm for cutwidth.
(4) Are there practically relevant, special graph classes where bandwidth is fixed parameter tractable (with a "reasonable" running time)?
(5) Dive into the ocean of VLSI design literature and find new candidates for parameterized complexity studies, e.g., one candidate might be the via minimization problem and its several variants [21,26,71,86].

5 Computational biology problems

Computational biology, a field still in its infancy, seems to contain a lot of problems suitable for parameterized complexity studies. A very algorithmic area of research within computational biology is phylogenetics, in which an important goal is to construct an evolutionary tree (also called *phylogenetic* tree) for a given set of species. Besides the context of phylogenetics, where we consider the problems perfect phylogeny, MAST, and gene duplication, we also present examples from two other areas: From genome rearrangement, dealing with the comparison of genomes having a known set of genes, and from the comparison of RNA and protein sequences, in which an essential question is to incorporate the three-dimensional structure into the models.

5.1 Some results

Perfect phylogeny: Agarwala and Fernández-Baca [2,34] approach the question of building a phylogenetic tree for a given set of species in the following model. For a given set C of m characters they allow a character $c \in C$ to take one state of a fixed set of character states A_c. These characters may, e.g., represent properties of single organisms or the positions in its DNA sequence with the nucleotide bases being the character states. In this setting, we are given a set S of n species, for which we intend to construct a tree. Each species $s \in S$ is represented by a vector of character states $s \in A_1 \times \cdots \times A_m$. The perfect phylogeny problem is then to determine whether there is a tree T with nodes $V(T) \subseteq A_1 \times \cdots \times A_m$, where each leaf of the tree is a species. In addition, we require for every $c \in C$ and every $j \in A_c$, the set of all nodes u of the tree with u's character c being in state j to induce a subtree. Downey and Fellows [34] refer to this problem as bounded character state perfect phylogeny. This indicates that the parameter here is the maximum number of character states $r = \max_{c \in C}|A_c|$. Using a dynamic programming approach and building perfect phylogenies from bottom up, Agarwala and Fernández-Baca present an $O(2^{3r}(nm^3 + m^4))$ time algorithm, which was improved by Kannan and Warnow to $O(2^{2r}nm^2)$ [63].

A generalization of PERFECT PHYLOGENY is the l-PHYLOGENY problem in which the question is to construct a tree T such that, given the fixed integer l, each character state does not induce more than l connected components in T. A parameterized analysis of the problem was started by Goldberg *et al.* [49] (also see for more details on the definition).

MAST: For a given set of species, we may obtain several phylogenetic trees, e.g., by building trees based on different gene families. These so-called *gene trees* are a good hypothesis for a *species tree*, i.e., the evolutionary relationship of the species, if they are all the same. If, however, the trees do not agree for all species, the following problem tries to find those species for which they do. Given a set of rooted trees $T_1, \ldots, T_r$ with the leaf set of each T_i labeled one-to-one with the set of species X and given a positive integer k, the MAXIMUM AGREEMENT SUBTREE PROBLEM (MAST) is to determine whether there is a subset $S \subseteq X$ of size at most k such that all trees T_i restricted to the leaf set $X - S$ are the same (up to isomorphism and ignoring nodes of degree two) for $i = 1, \ldots, r$. Therefore, the parameter k is the number of species we are willing to exclude from our analysis.

For three input trees of unbounded degree the problem is NP-hard [5]. For input trees with degree bounded by a constant d, it is solvable in polynomial time $O(rn^3 + n^d)$ [39]. A fixed parameter algorithm can be obtained as follows. Bryant [18] proposes to consider triples of species and, for each triple, the possible topologies for a rooted tree with three leaves. When several of these topologies arise among the given input trees, the triple is called *conflicted* and at least one of the three species has to be deleted and to be in the set S. Using this idea, Downey *et al.* [36] give an $O(3^k rn \log n)$ time solution for binary input trees. Independently of the input trees' degree, we can observe that a solution exists iff there is a set, namely S, of at most k species such that each conflicted triple contains one of these species. In this way, MAST can be reduced to the 3-HITTING SET problem [19,36]. Here, the problem is, given a collection C of subsets of size three of a finite set X and a positive integer k, to determine a subset $S \subseteq X$ with $|S| \leq k$ so that S contains at least one element from each subset in C. This problem is *NP*-complete. With a recent algorithm for 3-HITTING SET [74] using bounded search trees and reduction to problem kernel, the MAST problem can be solved in time $O(2.270^k + rn^3)$.

Gene duplication: As we have seen in the previous problem, when considering several gene families for a set of species, the gene trees can differ from the species tree. A way to explain the contradictions in the trees is the possibility that genes are duplicated in the evolutionary history and evolve independently. This observation motivates the following model to infer a species tree from several, possibly contradictory gene trees, the GENE DUPLICATION problem: Given a set of species and a set of trees (the gene trees) with their leaves labeled from the species set, the question is, intuitively speaking, to find a tree (the species tree) that requires a minimum number of gene duplications in order to explain the given gene trees (refer to [41,88] for further details). Note that in this model we count duplication events copying only one gene at a time. Stege [88] gives a fixed parameter algorithm for GENE DUPLICATION, with the allowed number of duplications as the parameter.

As a duplication event in evolutionary history copies a piece of DNA with possibly many genes on it, Fellows *et al.* [41] study the MULTIPLE GENE DUPLICATION problem. In contrast to GENE DUPLICATION, here, one duplication event copies a set of genes. With the upper bound on the number of duplications as parameter, they show even the easier version to be $W[1]$-hard where we are also given the species tree and only ask for the minimum number of required duplications.

Genome Rearrangement: Knowing the succession of genes on a chromosome, a way to measure the similarity of two corresponding chromosomes from different organisms with the same set of genes, is to count the number of mutation events required to obtain one succession of genes from the other. Examples of such mutation events are, e.g., inverting a subsequence, called *reversal*, or their deletion and insertion at another position, called *transposition*. Reversals are the most common kind of these mutations. Restricting to them, the comparison of two sequences of the same set of genes is modeled in the SORTING BY REVERSALS problem: Given a permutation π of $\{1, 2, \ldots, n\}$, the question is to find the minimum number of reversals we need to transform π into *id*. SORTING BY REVERSALS is *NP*-complete [23]. Hannenhalli and Pevzner's results [56], however, imply a fixed parameter algorithm for the problem when parameterized by the number of reversals.

Another genome-level distance measure that was shown to be fixed parameter tractable is the SYNTENIC DISTANCE [43]. In this model, a genome is given by k subsets of a set of n genes. These subsets represent the chromosomes and the elements in a set represent the genes located on the chromosome. The mutation events in this model are the union of two chromosome sets, the splitting of a chromosome set into two sets, and the exchange of genes between two sets. Given two genomes G_1 and G_2, the SYNTENIC DISTANCE problem is to compute the minimum number of mutation events needed to transform G_1 into G_2. DasGupta *et al.* [30] show that computing the SYNTENIC DISTANCE is *NP*-hard and fixed parameter tractable when parameterized by the distance.

Comparison of RNA and protein structure: The three-dimensional structure of RNA or protein sequences is considered to be important for their function and for the degree of their evolutionary and functional similarity. Therefore, it is important to find models which do incorporate this structure when comparing sequences. An approach common for RNA sequences is to represent bonds between single bases by arcs between positions in the sequence, leading to so-called *annotated* sequences. Considering these arcs when comparing two sequences, their longest common subsequence can be interpreted as a measure for similarity not only on the sequence level, but also on the structural level. Evans [37] proposes the following model, which takes as input two annotated sequences S_1 and S_2 of length n and m, respectively. Thereby, crossing edges are allowed, in order to represent three-dimensional structures. Further inputs are an integer l denoting the target length and an integer k denoting the cutwidth, i.e., the maximum number of arcs crossing a position. The ARC-PRESERVING LONGEST COMMON SUBSEQUENCE problem is, in this context, to determine whether there is a common subsequence in S_1 and S_2, which preserves the arcs, is of length at least l and has cutwidth at most k. Evans points out that with different parameterizations the problem exhibits different parameterized complexity. It is $W[1]$-complete when the parameter is the target length l and the cutwidth is not limited. If, however, we take the cutwidth k as the parameter without having a target length, the problem is fixed parameter tractable: the algorithm for this variant finds the longest common subsequence in time $O(9^k nm)$ [37].

A similar model to compare the three-dimensional structure of proteins is the CONTACT MAP OVERLAP problem described by Goldman *et al.* [50]. They give results showing *MaxSNP*- and *NP*-hardness of the problem and identify special versions relevant in practice that are solvable in polynomial time by dynamic programming. A parameterized analysis of the problem is open.

5.2 Some challenges

(1) Can the parameterized analysis of l-PHYLOGENY started by Goldberg *et al.* [49] be continued?
(2) Can the fixed parameter algorithm solving SORTING BY REVERSALS with the distance as parameter be extended to also allow other mutation events, e.g., transpositions and duplications?
(3) The analysis of protein structure is a tremendously important issue in computational biology. With respect to the question of comparing two proteins, can we study the parameterized complexity of CONTACT MAP OVERLAP [50] and identify parameters that make the problem tractable?
(4) In Section 3.1, we mentioned the PROPER INTERVAL GRAPH COMPLETION problem relevant in physical mapping. The problem asks whether a given graph can be turned into a proper interval graph (defined in [64]) by adding at most k edges and is shown to be fixed parameter tractable [64] for parameter k. What is the parameterized complexity of the problem

when we allow removing edges, adding vertices, and/or removing vertices?

(5) The optimal alignment of two sequences [55] can usually be done in polynomial time. The alignment of an arbitrary number of sequences is usually NP-complete. In both cases we say "usually," as these results depend on the model and the used function to score an alignment. It would be interesting to survey the parameterized complexity of multiple alignment in different models.

6 Logic problems

Logic, seen in a broad sense, is another field of diverse problems to be attacked with parameterized complexity methods. We basically present three types of logic problems: the optimization problem MAXIMUM SATISFIABILITY, the "classical logic" problem FALSIFIABILITY FOR PURE IMPLICATIONAL FORMULAS, and the complexity of DATABASE QUERIES.

6.1 Some results

Maximum satisfiability: This is a problem especially well-known from the field of approximation algorithms [1,60] and heuristic methods [11], having also important practical applications [57]. The input instance is a boolean formula in conjunctive normal form (CNF), and the problem is to find a truth assignment that satisfies the maximum number of clauses. The decision version of MAXIMUM SATISFIABILITY is NP-complete, even if the clauses have at most two literals (so-called MAXIMUM 2-SATISFIABILITY) [48]. One of the major results in theoretical computer science in recent times shows that if there is a polynomial time approximation scheme for MAXIMUM SATISFIABILITY, then $P = NP$ [7].

The natural parameterized version of MAXIMUM SATISFIABILITY requires an algorithm to determine whether at least k clauses of a CNF formula F can be satisfied. Assume that F contains m clauses and n variables. For each F, there always exists a truth assignment satisfying at least $\lceil m/2 \rceil$ clauses: simply pick any assignment—either it does or its bitwise complement does. This can be checked in time $O(|F|)$. For this reason, Mahajan and Raman [67] introduced a more meaningful parameterization, asking whether at least $\lceil m/2 \rceil + k$ clauses of a CNF formula F can be satisfied. However, the first parameterization still remains of interest since, from a "non-parameterized point of view," an algorithm with running time exponential in m with a small base for the exponential factor can be of interest. Thus, we firstly stick to this basic parameterization and afterwards very briefly deal with the "more meaningful" parameterization.

Mahajan and Raman [67] presented an algorithm running in time $O(|F| + 1.6181^k k^2)$ that determines whether at least k clauses of a CNF formula F are satisfiable. This algorithm uses a reduction to problem kernel as well as a bounded search tree. The reduction to the problem kernel relies on the distinction between "large" clauses (i.e., clauses containing at least k literals) and "small" clauses (i.e., clauses containing less than k literals). If F contains at least k large clauses, then it is easy to see that at least k clauses in F can be satisfied. Hence, the subsequent search tree method has only to deal with small clauses. Observe that the size of the remaining "subformula of small clauses" can easily be bounded by $O(k^2)$. This is also owing to the fact that, if the number of clauses in F is at least $2k$, then, trivially, k clauses in F can be satisfied. Now the bounded search tree, here more appropriately called branching tree, appears as follows. First, note that we can restrict ourselves to only considering variables that occur both positively and negatively in F, because so-called "pure literals" can always be set true, always increasing the number of satisfied clauses without any disadvantage. The basic technique now is to pick one variable x occurring both positively and negatively in F and then to "branch" into two subformulas $F[x]$ and $F[\bar{x}]$, which arise by setting x to true and false. Clearly, the size of such a branching tree can easily be bounded by 2^k. However, Mahajan and Raman [67] use a further trick based on "resolution," leading to the above running time. Using many more, carefully designed transformation and splitting rules for propositional formulas, the above result could be improved to time complexity $O(|F| + 1.3995^k k^2)$ [76] and, based on this, even further to $O(|F| + 1.3803^k k^2)$ [10]. We only mention in passing that the parameterization of MAXIMUM SATISFIABILITY requiring the satisfiability of at least $\lceil m/2 \rceil + k$ clauses is led back to the case considered above by Mahajan and Raman, thus obtaining a time complexity of $O(|F| + 1.6181^{6k} k^2) \approx O(|F| + 17.9443^k k^2)$. Plugging in the above described improvements [10], we immediately get $O(|F| + 1.3803^{6k} k^2) \approx O(|F| + 6.9158^k k^2)$.

The special case of MAXIMUM SATISFIABILITY where each clause may contain at most 2 literals has received special attention [51,52]. Among other things, this is due to the fact that important *NP*-complete graph problems such as MAXIMUM CUT and INDEPENDENT SET can be reduced to special instances of MAXIMUM 2-SATISFIABILITY [25,67]. Not surprisingly, the upper bounds for MAXIMUM 2-SATISFIABILITY are better than those for the general case, for the time being, however, parameterized complexity studies seem to fail in this context: In time $L^{O(1)} \cdot 2^{m_2/5}$ one can determine a maximum satisfying assignment, where L is the total number of literal occurrences in the formula and m_2 is the number of clauses of size two occurring in it [51]. This also holds for clauses with positive integer weights. In an earlier paper [52], the parameterized bound $2^{k/2.73}$ for MAXIMUM 2-SATISFIABILITY has been proved. However, the above "unparameterized" bound $2^{m_2/5}$, where m_2 is the number of 2-clauses, is better for all reasonable values of k: the parameterized bound is better only when $k < 0.55m_2$, while an assignment satisfying $0.5m + 0.25m_2 \geq 0.75m_2$ can be found in polynomial time [67]. It seems like the idea of counting only 2-clauses does not work for parameterized bounds.

Falsifiability problem for pure implicational formulas: The complexity of this problem was first studied by Heusch [59]. A Boolean formula is in pure implicational form if it contains only positive literals and the only logical connective being used is the implication. Heusch considered the special case when all variables except at most one (denoted z) occur at most twice. This problem still is NP-complete [59]. However, he proved that if the number of occurrences of z is restricted to be at most k, then there is an $O(|F|^k)$ time algorithm for certifying falsifiability. Franco *et al.* [46] subsequently showed how to solve the FALSIFIABILITY PROBLEM FOR PURE IMPLICATIONAL FORMULAS in time $O(k^k n^2)$; thus, this problem is fixed parameter tractable.

Database problems: Finally, let us take a very brief look at connections between parameterized complexity and database problems, which we boldly term as logic problems: Papadimitriou and Yannakakis revisited the complexity of DATABASE QUERIES in the light of parameterized complexity [81]. Here, the basic observation is that the size of the queries is typically orders of magnitude smaller than the size of the database. They analyze the complexity of the queries (e.g., conjunctive queries, first-order, Datalog, fixpoint logic etc.) with respect to two types of parameters: the query size itself and the number of variables that appear in the query. In this setting, they classify the relational calculus and its fragments at various levels of the W-hierarchy, hence showing parameterized intractability. On the positive side, they show that the extension of acyclic queries with inequalities is fixed parameter tractable (refer to [81] for details). Finally, let us only mention in passing that also in the field of model checking parameterized complexity studies are useful [45,66].

6.2 Some challenges

(1) A "non-parameterized" challenge: Can MAXIMUM 2-SATISFIABILITY or even MAXIMUM SATISFIABILITY be solved in less than 2^n "steps?" Here, n denotes the number of different variables in the formula and a step may take polynomial time.
(2) Can the results for MAXIMUM SATISFIABILITY be generalized to so-called MAXIMUM CONSTRAINT SATISFACTION PROBLEMS (cf., e.g., [12])?

(3) Is there a direct way to solve the "more meaningful parameterization" [67] of MAXIMUM SATISFIABILITY more efficiently than by just reducing it to the standard parameterized version?
(4) What are the best (parameterized) bounds for MAXIMUM CUT? We only have the bounds derived from reductions to MAXIMUM 2-SATISFIABILITY.
(5) Can the time bound $O(k^k n^2)$ for the FALSIFIABILITY PROBLEM FOR PURE IMPLICATIONAL FORMULAS WITH k NEGATIONS be further improved?

7 And what about experimental results?

There are numerous examples of NP-complete problems for which theoretical upper bounds have been shown on the running time. However, these results are only rarely accompanied by an implementation. In most cases, it remains open as to how these algorithms behave in practice. Since better theoretical results often rely on more complex constant-time computations, it is reasonable that practitioners ask whether these strategies are still efficient and how, e.g., they compare to known heuristics or how they can be combined with heuristic approaches.

In the following, we concretely describe some experiments made with algorithms designed for the MAXIMUM 2-SATISFIABILITY problem [52,53] and try to draw some general conclusions. The algorithms use the strategy of bounded search trees, as explained in Section 6.1. Bounded search trees are a common pattern of fixed parameter algorithms. Therefore, we conjecture that the following observations are typical for these kinds of algorithms. Note, however, that the test runs are done with the non-parameterized version of the algorithm, i.e., the algorithm which searches an assignment satisfying the maximum number of clauses. For a given parameter k, the parameterized version would stop as soon as k clauses are satisfied. Apart from this, the algorithms work analogously, such that the same observations apply for the parameterized as for the non-parameterized version.

Before we start to outline our experiences, we quickly review the implemented strategy. The algorithms traverse a search tree of exponential size. In each node of this tree, the current problem instance is simplified by excluding subcases which give no improvement of the maximal number of satisfied clauses, e.g., by assigning values to pure literals. The known simplifications of this kind are manifested in a set of seven *transformation rules*, which are applied if possible. As soon as none of these transformation rules applies any more, we branch into two or more subcases by setting selected variables to certain values, e.g., setting a variable to true in one case and false in the other, and work on each of these subcases recursively. The subcases to be branched into are determined by a set of eight *branching rules*. The involved analysis of these branching cases allows to give estimations for the size of the search tree. The questions resulting from this strategy are whether it is efficient to test for transformation and branching rules and to maintain the quite complex data structures necessary in order to decide which of the rules applies. In order to answer these questions, the algorithms were implemented in JAVA and tested on a common Linux-PC with an AMD K6 processor with 233 MHz. In the following, we outline some important observations resulting from these experiments:

In order to judge their performance, we compared the algorithms with the heuristic EDPL (Extended Davis-Putnam-Loveland) strategy presented by Borchers and Furman [17], for which an implementation in C is publicly available. For all problem instances, the new algorithms produced smaller search trees than the EDPL strategy. Regarding the running time, the new algorithms were outperformed by the EDPL strategy on instances of very small size despite the larger search trees of the latter. One reason is certainly the performance difference of the programming languages (C was measured to be faster than JAVA in this setting by a factor of about 9). But with growing instances, the algorithms are faster than the EDPL strategy, with an exponentially growing gap between them. Instances handled by the new algorithms in seconds, e.g., random formulas with 200 variables in 400 clauses, cannot be processed by the EDPL strategy in reasonable time. In this case, the additional time the new algorithms spend in each search tree node, e.g., to maintain the data structures and to perform the case analysis, is definitely outweighed by the gain through the shrunken search trees.

Although the algorithms exhibited a good performance in general, we encountered examples in which better theoretical bounds did not imply faster algorithms. E.g., we tested our algorithms not only with randomly generated 2-CNF formulas but, additionally, we derived formulas from maximum cut instances, which can easily be reduced to maximum 2-satisfiability. As these maximum cut instances exhibit a special structure, the theoretical analysis can be improved for them, yielding better upper bounds [51]. In practice, however, the maximum cut instances turn out to be harder for the algorithms than random instances.

Besides the overall performance, we were interested in the role of single transformation and branching rules. Therefore, we collected statistical data about the application of these rules. Regarding the transformation rules, we note the high number of applications. Regarding the branching rules, we observe that most of the branching rules are used only in rare cases. Most branchings result from two of the eight rules. This indicates that the carefully designed branching rules mainly serve for the theoretical analysis which gives guarantees on the upper bounds.

In conclusion, we point out the following experiences, which seem to be important for all algorithms using a similar kind of strategy:

- The algorithms are able to exhibit a good performance comparable to current exact heuristics. Note that it is possible to enrich and improve the algorithms with heuristic strategies, e.g., the branch and bound principle, without sacrificing the proven performance bounds.
- There are examples showing that an improvement of theoretical bounds does not necessarily imply faster algorithms.
- There is a difference in the roles of transformation and branching rules. Therefore, to improve the practical performance, it seems to be more promising to search for further transformation rules. Along with the set of transformation rules, a further refinement of branching rules promises better theoretical upper bounds.

8 Conclusion

To study faster exact solutions for *NP*-hard problems, one always should check whether a parameterized point of view (also) makes sense. A parameterized approach may help "improving" known upper bounds; for example, this holds true in the case of VERTEX COVER (compare the parameterized [24] with the non-parameterized [84] results). We tried to illustrate that parameterized problems appear almost everywhere—we gave a few examples from graph theory, VLSI design, computational biology, and logic. The study of whether or not a problem is fixed parameter tractable and, if yes, how small the exponential terms can be made, might perhaps deserve similar attention as questions for the approximability of problems currently obtain.

In this paper, we focused on concrete examples and concrete open problems concerning the development of (efficient) fixed parameter algorithms. To some extent, we neglected more general, probably "more structural" open problems in parameterized complexity analysis, which, nevertheless, should play a major role in future research. This includes questions for the relationship between approximation and parameterized complexity, the relationship between heuristics and parameterized complexity, and a closer investigation of algorithmic paradigms as linear or integer programming or dynamic programming in relation with parameterized complexity. Many challenges remain.

Acknowledgment. We thank Mike Fellows, Henning Fernau, Mike Hallett, Ton Kloks, Klaus-Jörn Lange, Peter Rossmanith, and Ulrike Stege for contributing to our knowledge on parameterized complexity. In particular, the third author is grateful to Mike Fellows, Jan Kratochvíl, Jarik Nešetřil, and

Jiří Wiedermann for "pushing" him into this field of research in one way or another.

References

[1] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation.* Springer-Verlag Berlin Heidelberg, 1999.

[2] R. Agarwala and D. Fernández-Baca. A polynomial-time algorithm for the perfect phylogeny problem when the number of character states is fixed. *SIAM Journal on Computing*, 23(6):1216–1224, 1994.

[3] J. Alber, H. L. Bodlaender, H. Fernau and R. Niedermeier. Fixed parameter algorithms for planar dominating set and related problems. In *Proceedings of the 7th Scandinavian Workshop on Algorithm Theory*, number 1851 in Lecture Notes in Computer Science, pages 97–110. Springer-Verlag, July 2000.

[4] N. Alon, R. Yuster, and U. Zwick. Color-coding. *Journal of the ACM*, 42(4):844–856, 1995.

[5] A. Amir and D. Keselman. Maximum agreement subtrees in multiple evolutionary trees. In *Proceedings of the 35th IEEE Symposium on Foundations of Computer Science*, pages 758–769, 1994.

[6] S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Alg. Disc. Meth.*, 8:277–284, 1987.

[7] S. Arora and C. Lund. Hardness of approximation. In D. Hochbaum, editor, *Approximation algorithms for NP-hard problems*, chapter 10, pages 399–446. PWS Publishing Company, Boston, 1997.

[8] B. S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM* 41:153–180, 1994.

[9] R. Balasubramanian, M. R. Fellows, and V. Raman. An improved fixed parameter algorithm for vertex cover. *Information Processing Letters*, 65(3):163–168, 1998.

[10] N. Bansal and V. Raman. Upper bounds for MaxSat: Further improved. In *Proceedings of the 10th International Symposium on Algorithms and Computation*, number 1741 in Lecture Notes in Computer Science, pages 247–258. Springer-Verlag, December 1999.

[11] R. Battiti and M. Protasi. Approximate algorithms and heuristics for MAX-SAT. In D.-Z. Du and P. M. Pardalos, editors, *Handbook of Combinatorial Optimization*, volume 1, pages 77–148. Kluwer Academic Publishers, 1998.

[12] R. Battiti and M. Protasi. Reactive local search techniques for the maximum k-conjunctive constraint satisfaction problem (MAX-k-CCSP). *Discrete Applied Mathematics*, 96–97:3–27, 1999.

[13] A. Blummer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, 36:929–965, 1989.

[14] H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25:1305–1317, 1996.

[15] H. L. Bodlaender. Treewidth: Algorithmic techniques and results. In *Proceedings of the 22nd Conference on Mathematical Foundations of Computer Science*, number 1295 in Lecture Notes in Computer Science, pages 19–36. Springer-Verlag, 1997.

[16] R. B. Boppana and M. Sipser. The complexity of finite functions. In J. van Leeuwen, editor, *Algorithms and Complexity*, volume A of *Handbook of Theoretical Computer Science*, chapter 14, pages 757–804. Elsevier, 1990.

[17] B. Borchers and J. Furman. A two-phase exact algorithm for MAX-SAT and weighted MAX-SAT problems. *Journal of Combinatorial Optimization*, 2(4):299–306, 1999.

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

[19] D. Bryant, M. Fellows, V. Raman, and U. Stege. On the parameterized complexity of MAST and 3-Hitting Sets. Unpublished manuscript, 1998.

[20] N. H. Bshouty and L. Burroughs. Massaging a linear programming solution to give a 2-approximation for a generalization of the vertex cover problem. In *Proceedings of the 15th Symposium on Theoretical Aspects of Computer Science*, number 1373 in Lecture Notes in Computer Science, pages 298–308. Springer-Verlag, 1998.

[21] T. N. Bui, W. Hsu, and S.-S. Lee. A 2.5 approximation algorithm for the multi-via assignment problem. *IEEE Transactions on Computer-Aided Design*, 11(11):1325–1333, 1992.

[22] L. Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 58:171–176, 1996.

[23] A. Caprara. Sorting permutations by reversals and Eulerian cycle decompositions. *SIAM Journal on Discrete Mathematics*, 12:91–110, 1999.

[24] J. Chen, I. Kanj, and W. Jia. Vertex cover: Further observations and further improvements. In *Proceedings of the 25th International Workshop on Graph-Theoretic Concepts in Computer Science*, number 1665 in Lecture Notes in Computer Science, pages 313–324. Springer-Verlag, June 1999.

[25] J. Cheriyan, W. H. Cunningham, L. Tunçel, and Y. Wang. A linear programming and rounding approach to Max 2-Sat. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 26:395–414, 1996.

[26] H.-A. Choi, K. Nakajima, and C. S. Rim. Graph bipartization and via minimization. *SIAM J. Disc. Math.*, 2(1):38–47, 1989.

[27] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetical progression. *J. Symbolic Computations*, 9:251–280, 1990.

[28] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. Introduction to Algorithms. The MIT Press, 1990.

[29] P. Crescenzi and V. Kann. A compendium of NP optimization problems. Available at http://www.nada.kth.se/theory/problemlist.html, August 1998.

[30] B. DasGupta, T. Jiang, S. Kannan, M. Li, and E. Sweedyk. On the complexity and approximation of syntenic distance. *Discrete Applied Mathematics*, 88:59–82, 1998.

[31] R. Diestel. *Graph Theory.* Springer-Verlag, New York, 1997.

[32] R. G. Downey, P. Evans, and M. R. Fellows. Parameterized learning complexity. In *6th Annual Conference on Learning Theory, COLT'93*, pages 51–57. ACM Press, 1993.

[33] R. G. Downey and M. R. Fellows. Parameterized computational feasibility. In *P. Clote, J. Remmel (eds.): Feasible Mathematics II*, pages 219–244. Boston: Birkhäuser, 1995.

[34] R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer-Verlag, 1999.

[35] R. G. Downey and M. R. Fellows. Parameterized complexity after (almost) ten years: Review and open questions. In *Combinatorics, Computation & Logic, DMTCS'99 and CATS'99*, Australian Computer Science Communications, Volume 21 Number 3, pages 1–33. Springer-Verlag Singapore, 1999.

[36] R. G. Downey, M. R. Fellows, and U. Stege. Parameterized complexity: A framework for systematically confronting computational intractability. In *Contemporary Trends in Discrete Mathematics: From DIMACS and DIMATIA to the Future*, volume 49 of *AMS-DIMACS*, pages 49–99. AMS, 1999.

[37] P. A. Evans. Finding common subsequences with arcs and pseudoknots. In *Proceedings of the 10th Annual Symposium Combinatorial Pattern Matching (CPM)*, number 1645 in Lecture Notes in Computer Science, pages 270–280. Springer-Verlag, 1999.

[38] R. C. Evans. Testing repairable RAMs and mostly good memories. In *Proceedings of the IEEE Int. Test Conf.*, pages 49–55, 1981.

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

[40] U. Feige. Coping with the *NP*-hardness of the graph bandwidth problem. In *Proceedings of the 7th Scandinavian Workshop on Algorithm Theory*, number 1851 in Lecture Notes in Computer Science, pages 10–19. Springer-Verlag, July 2000.

[41] M. Fellows, M. Hallett, and U. Stege. On the multiple gene duplication problem. In *Proceedings of the 9th International Symposium on Algorithms and Computation*, number 1533 in Lecture Notes in Computer Science, pages 347–356. Springer-Verlag, December 1998.

[42] M. R. Fellows and M. A. Langston. On well-partial-ordering theory and its applications to combinatorial problems in VLSI design. *SIAM J. Discrete Math.*, 5:117–126, 1992.

[43] V. Feretti, J. H. Nadeau, and D. Sankoff. Original synteny. In *Proceedings of the 7th Annual Symposium Combinatorial Pattern Matching (CPM)*, number 1075 in Lecture Notes in Computer Science, pages 159–167. Springer-Verlag, 1996.

[44] H. Fernau and R. Niedermeier. An efficient exact algorithm for Constraint Bipartite Vertex Cover. In *Proceedings of the 24th Conference on Mathematical Foundations of Computer Science*, number 1672 in Lecture Notes in Computer Science, pages 387–397. Springer-Verlag, September 1999.

[45] J. Flum and M. Grohe. Fixed-parameter tractability and logic. *Technical Report [99-23]*, Mathematische Fakultät, Albert-Ludwigs-Universität Freiburg, 1999.

[46] J. Franco, J. Goldsmith, J. Schlipf, E. Speckenmeyer, and R.. P. Swaminathan. An algorithm for the class of pure implicational formulas. *Discrete Applied Mathematics*, 96-97:89–106, 1999.

[47] U. Fößmeier and M. Kaufmann. On exact solutions for the rectilinear Steiner tree problem—part I: Theoretical results. *Algorithmica*, 26:68–99, 2000.

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

[49] L. A. Goldberg, P. W. Goldberg, C. A. Phillips, E. Sweedyk, and T. Warnow. Minimizing phylogenetic number to find good evolutionary trees. *Discrete Applied Mathematics*, 71(1–3):111–136, 1996.

[50] D. Goldman, S. Istrail, and C. H. Papadimitriou. Algorithmic aspects of protein structure similarity. In *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science*, pages 512–521, 1999.

[51] J. Gramm, E. A. Hirsch, R. Niedermeier, and P. Rossmanith. New worst-case upper bounds for MAX-2-SAT with application to MAX-CUT. Invited for submission to a special issue of *Discrete Applied Mathematics*, May 2000. Also appears as ECCC Technical Report TR00-037, Trier, Fed. Rep. of Germany.

[52] J. Gramm and R. Niedermeier. Faster exact solutions for Max-2-Sat. In *Proceedings of the 4th Italian Conference on Algorithms and Complexity*,

number 1767 in Lecture Notes in Computer Science, pages 174–186. Springer-Verlag, March 2000.

[53] J. Gramm. Exact algorithms for Max2Sat and their applications. Diploma thesis, WSI für Informatik, Universität Tübingen, 1999. Available through http://www-fs.informatik.uni-tuebingen.de/~gramm/publications/

[54] M. Grohe. Descriptive and parameterized complexity. In *Computer Science Logic, 13th Workshop*, number 1683 in Lecture Notes in Computer Science, pages 14–31. Springer-Verlag, September 1999.

[55] D. Gusfield. *Algorithms on Strings, Trees, and Sequences (Computer Science and Computational Biology).* Cambridge University Press, 1997.

[56] S. Hannenhalli and P. Pevzner. To cut ... or not to cut (applications of comparative physical maps in molecular evolution). In *Proceedings of the 7th ACM-SIAM Symposium on Discrete Algorithms*, pages 304–313, 1996.

[57] P. Hansen and B. Jaumard. Algorithms for the maximum satisfiability problem. *Computing*, 44:279–303, 1990.

[58] N. Hasan and C. L. Liu. Fault covers in reconfigurable PLAs. In *20th Int. Symp. on Fault-Tolerant Computing Systems (FTCS'90)*, pages 166–173. IEEE Computer Society Press, 1990.

[59] P. Heusch. The complexity of the falsifiability problem for pure implicational formulas. *Discrete Applied Mathematics*, 96-97:127–138, 1999.

[60] D. S. Hochbaum, editor. *Approximation algorithms for NP-hard problems.* Boston, MA: PWS Publishing Company, 1997.

[61] D. S. Hochbaum. The t-vertex cover problem: Extending the half integrality framework with budget constraints. In *Proceedings of APPROX98*, number 1444 in Lecture Notes in Computer Science, pages 111–122, July 1998. Springer-Verlag.

[62] M. Hofri. *Analysis of algorithms: computational methods and mathematical tools.* Oxford University Press, 1995.

[63] S. Kannan and T. Warnow. A Fast algorithm for the computation and enumeration of perfect phylogenies. *SIAM Journal on Computing*, 26(6):1749–1763, 1997.

[64] H. Kaplan, R. Shamir, and R. E. Tarjan. Tractability of parameterized completion problems on chordal, strongly chordal, and proper interval graphs. *SIAM Journal on Computing*, 28(5):1906–1922, 1999.

[65] S.-Y. Kuo and W.K. Fuchs. Efficient spare allocation for reconfigurable arrays. *IEEE Design and Test*, 4:24–31, February 1987.

[66] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their specification. In *12th Annual ACM Symp. on Principles of Prog. Lang.*, pages 97–107, 1985.

[67] M. Mahajan and V. Raman. Parameterizing above guaranteed values: MaxSat and MaxCut. *Journal of Algorithms*, 31:335–354, 1999.

[68] K. Mehlhorn. *Graph algorithms and NP-completeness.* Springer-Verlag, 1984.

[69] Z. Michalewicz and D. B. Fogel. *How to solve it: Modern Heuristics.* Springer-Verlag, 2000.

[70] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[71] N. J. Naclerio, S. Masuda, and K. Nakajima. The via minimization problem is NP-complete. *IEEE Trans. on Computers*, 38(11):1604–1608, 1989.

[72] G. L. Nemhauser and L. E. Trotter Jr. Vertex packings: Structural properties and algorithms. *Mathematical Programming*, 8:232–248, 1975.

[73] J. Nešetřil and S. Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 26(2):415–419, 1985.

[74] R. Niedermeier and P. Rossmanith. An efficient fixed parameter algorithm for 3-Hitting Set. Technical Report WSI-99-18, WSI für Informatik, Universität Tübingen, Fed. Rep. of Germany, October 1999. Revised version to appear in *Journal of Discrete Algorithms*.

[75] R. Niedermeier and P. Rossmanith. A general method to speed up fixed-parameter-tractable algorithms. *Information Processing Letters*, 73:125–129, 2000.

[76] R. Niedermeier and P. Rossmanith. New upper bounds for Maximum Satisfiability. *Journal of Algorithms*, 36: 63–88, 2000.

[77] R. Niedermeier and P. Rossmanith. Upper bounds for Vertex Cover further improved. In *Proceedings of the 16th Symposium on Theoretical Aspects of Computer Science*, number 1563 in Lecture Notes in Computer Science, pages 561–570. Springer-Verlag, 1999.

[78] R. Niedermeier and P. Rossmanith. On efficient fixed parameter algorithms for Weighted Vertex Cover. In *Proceedings of the 11th International Symposium on Algorithms and Computation*, Lecture Notes in Computer Science, Taipei, Taiwan. Springer-Verlag, December 2000, to appear.

[79] C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[80] C. H. Papadimitriou and M. Yannakakis. On limited nondeterminism and the complexity of the V-C dimension. *Journal of Computer and System Sciences*, 53:161–170, 1996.

[81] C. H. Papadimitriou and M. Yannakakis. On the complexity of database queries. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 12–19, 1997.

[82] V. T. Paschos. A survey of approximately optimal solutions to some covering and packing problems. *ACM Computing Surveys*, 29(2):171–209, June 1997.

[83] V. Raman. Parameterized complexity. In *Proceedings of the 7th National Seminar on Theoretical Computer Science (Chennai, India)*, pages I–1–I–18, June 1997.

[84] J. M. Robson. Algorithms for maximum independent sets. *Journal of Algorithms*, 7:425–440, 1986.

[85] J. P. Schmidt and A. Siegel. The spatial complexity of oblivious k-probe hash functions. *SIAM Journal on Computing*, 19(5):775–786, 1990.

[86] C.-J. Shi and J. A. Brzozowski. A characterization of signed hypergraphs and its applications to VLSI via minimization and logic synthesis. *Discrete Applied Mathematics*, 90:223–243, 1999.

[87] W. Shi and W. K. Fuchs. Probabilistic analysis of algorithms for reconfiguration of memory arrays. *IEEE Transactions on Computer-Aided Design*, 11(9):1153–1160, 1992.

[88] U. Stege. Gene trees and species trees: The gene-duplication problem is fixed-parameter tractable. In *Proceedings of the 6th Workshop on Algorithms and Data Structures*, number 1663 in Lecture Notes in Computer Science, pages 288–293. Springer-Verlag, 1999.

[89] U. Stege and M. Fellows. An improved fixed-parameter-tractable algorithm for vertex cover. Technical Report 318, Department of Computer Science, ETH Zürich, April 1999.

[90] J. A. Telle and A. Proskurowski. Practical algorithms on partial k-trees with an application to domination-like problems. In *Proceedings of the 3rd Workshop on Algorithms and Data Structures*, number 709 of Lecture Notes in Computer Science, pages 610–621. Springer-Verlag, 1993.

[91] J. A. Telle and A. Proskurowski, Algorithms for vertex partitioning problems on partial k-trees, *SIAM J. Discr. Math.*, 10(4):529–550, 1997.