What's next? Future directions in parameterized complexity

Dániel Marx*

Computer and Automation Research Institute
Hungarian Academy of Sciences (MTA SZTAKI)
Budapest, Hungary
dmarx@cs.bme.hu

Abstract. The progress in parameterized complexity has been very significant in recent years, with new research questions and directions, such as kernelization lower bounds, appearing and receiving considerable attention. This speculative article tries to identify new directions that might become similar hot topics in the future. First, we point out that the search for optimality in parameterized complexity already has good foundations, but lots of interesting work can be still done in this area. The systematic study of kernelization became a very successful research direction in recent years. We look at what general conclusions one can draw from these results and we argue that the systematic study of other algorithmic techniques should be modeled after the study of kernelization. In particular, we set up a framework for understanding which problems can be solved by branching algorithms. Finally, we discuss that the domain of directed graph problems is a challenging area which can potentially see significant progress in the following years.

1 Introduction

There was a guy whose name was Mike.
Loved math, surf, wine, and the like.
Once he climbed up a graph,
Took a photograph
And said: what a wonderful hike!

Zsuzsa Mártonffy

The field of parameterized complexity progressed enormously since the publication of Downey and Fellows' monograph [44] in 1999. New techniques and new discoveries opened up new research directions and changed the field, sometimes in unexpected ways. Kernelization, a basic algorithmic technique for obtaining fixed-parameter tractability results, has evolved into a subfield of its own by better understanding of its applicability and the possibility of proving strong

* Dedicated to Michael R. Fellows on the occasion of his 60th birthday. Research supported by the European Research Council (ERC) grant "PARAMTIGHT: Parameterized complexity and the search for tight complexity results," reference 280152.

upper and lower bounds. As explained by Langston elsewhere in this volume [73], the fact that the Graph Minors Theory of Robertson and Seymour (culminating in papers [94, 93]) implies the existence of polynomial-time algorithms in a nonconstructive way was one of the early motivations for parameterized complexity. In the past decade, an entirely different aspect of Graph Minors Theory has been developed, which allows us for example to generalize in many cases fixed-parameter tractability results from planar graphs to H-minor free graphs (see the survey of Thilikos in this volume [96]). A useful product of this development is the concept of bidimensionality, which changed substantially the way we look at planar graph problems [36–38]. Even simple techniques can move the field into new directions: iterative compression, introduced by Reed et al. [92], turned out to be a key step in proving the fixed-parameter tractability of important problems such as Bipartite Deletion [92], Almost 2SAT [91], and Directed Feedback Vertex Set [25], and changed the way we look at problems involving deletions.

One could list several other aspects in which the field evolved and changed in the past decade. However, the purpose of this article is *not* to review these developments. Rather than that, the purpose of this article is to propose some new directions for future research. Only time and further work can tell if these directions are as fruitful as the ones listed above.

The first topic we discuss is the optimality program of parameterized complexity: understanding quantitatively what is the best we can achieve for a particular problem. That is, instead of just establishing fixed-parameter tractability, we eventually want to understand the best possible $f(k)$ in the running time. This is not really a new direction, as the literature contains several results of this type. However, we feel that it is important to emphasize here that the search for optimality is a viable and very timely research program which should be guiding the development of the field in the following years.

Kernelization is perhaps the most practical technique in the arsenal of fixed-parameter tractability, thus it is not surprising that its methods and applicability have received particular attention in the literature. In the past few years, research on kernelization has increased enormously after it had been realized that the existence of polynomial kernels is a mathematically deep and very fruitful research question, both from the algorithmic and complexity points of view. A detailed overview of the results on kernelization is beyond the scope of this article; the reader is referred to [84, 12, 76] for a survey of recent results. However, we briefly review kernelization from the point of view of the optimality program. What we would like to point out is that the study of kernelization should be interpreted as a search for a tight understanding of the power of kernelization. That is, the question guiding our research is not which problems *can* be solved by kernelization, but rather which problems *should* be solved by kernelization.

Kernelization is just one technique in parameterized complexity and its systematic study opened up a whole new world of research questions. Could it be that exploring other basic techniques turns out to be as fruitful as the study of kernelization? Besides kernelization, branching is the most often used technique,

thus it could be the next natural target for rigorous analysis. We propose a framework in which one can study whether a problem can be solved by branching or not. Based on what we have learned from the study of kernelization, one should look at the study of branching also from the viewpoint of optimality: the goal is to understand for which problems is branching the right way of solution. The description and discussion of this framework is the only part of the paper containing new technical ideas. The presentation of this framework is intentionally kept somewhat informal, as going into the details of irrelevant technical issues would distract from the main message.

The last direction we discuss is the study of algorithmic problems on directed graphs. Perhaps it is premature to call such a wide area with disconnected results as a research direction. However, we would like to point out the enormous potential in pursuing questions in this direction. Problems on directed graphs are much more challenging than their undirected counterparts, as we are in a completely different world where many of the usual tools do not help at all. Still, there are directed problems that have been tackled successfully in recent years, for example, Directed Feedback Vertex Set [25] or Directed Multiway Cut [26]. This suggests that it is not hopeless to expect further progress on directed graphs, or even a general theory that is applicable for several problems.

2 The optimality program

Recall that a parameterized problem is *fixed-parameter tractable (FPT)* with a given parameterization if there is an algorithm with running time $f(k) \cdot n^{O(1)}$, where n is the size of the instance, k is the value of the parameter associated with the instance, and f is an arbitrary computable function depending only on the parameter k (see the monographs [44, 50, 87] or the survey [41] in this volume for more background). That is, the problem can be solved in polynomial time for every fixed value of the parameter k and the exponent *does not* depend on the parameter. Intuitively, we would like fixed-parameter tractability to express that the problem has an "efficient" or "practical" algorithm for small values of k. However, the definition only requires that f is computable and it can be any fast growing, ugly function. And this is not only a hypothetical possibility: the early motivation for parameterized complexity came from algorithmic results based on the Graph Minors Theory of Robertson and Seymour, and the $f(k)$ in these algorithms are typically astronomical towers of exponentials, far beyond any hope of practical use.

For most FPT problems, however, there are algorithms with "well-behaving" $f(k)$. In many cases, $f(k)$ is c^k for some reasonably small constant $c > 0$. For example, Vertex Cover can be solved in time $1.2738^k \cdot n^{O(1)}$ [23]. Sometimes the function $f(k)$ is even subexponential, e.g., $c^{\sqrt{k}}$. It happens very often that by understanding a problem better or by using more advanced techniques, better and better FPT algorithms are developed for the same problem and a kind of "race" is established to make $f(k)$ as small as possible. Clearly, it would be very useful to know if the current best algorithm can be improved further or it has

already hit some fundamental barrier. If a problem is NP-hard, then we cannot expect $f(k)$ to be polynomial. But is it possible that something very close to polynomial, say $k^{\log\log\log k}$ can be reached? Or are there problems for which the best possible $f(k)$ is very bad, say, of the form $2^{2^{2^{\Omega(k)}}}$? The optimality program tries to understand and answer such questions.

In recent years, a lower bound technology was developed, which, in many cases, is able to demonstrate the optimality of fixed-parameter tractability results. We sketch how such lower bounds can be proved using a complexity-theoretic assumption called Exponential Time Hypothesis (ETH). (An alternate way to discuss these results is via the complexity class M[1] and for some of the results even the weaker FPT $\neq$ W[1] hypothesis is sufficient. However, to keep our discussion focused, we describe only results based on ETH here.) For our purposes, ETH can be stated as follows:

Conjecture 2.1 (Exponential Time Hypothesis [63]) 3-SAT *cannot be solved in time* $2^{o(m)}$*, where* m *is the number of clauses.*

This conjecture was first formulated by Impagliazzo, Paturi, and Zane [63]. More precisely, they stated a version of the conjecture saying that there is no $2^{o(n)} \cdot m^{O(1)}$ time algorithm, where n is the number of variables, and showed by a reduction called the Sparsification Lemma that the two versions of the conjecture are equivalent. Although there is no universal consensus in accepting ETH (compared to more established conjectures such as P $\neq$ NP), it is consistent with our current knowledge: after several rounds of improvement, the best algorithm for n-variable m-clause 3-SAT has running time $O(1.30704^n)$ [60] and no algorithm with subexponential running time in m seems to be in sight.

If we accept ETH, then we can obtain lower bounds for other problems through reductions. Let us observe that standard NP-hardness reductions from 3-SAT to, say, Independent Set are sensitive to the number of clauses in the input instance. That is, there is a polynomial-time algorithm that, given an m-clause 3SAT instance ϕ, constructs a $O(m)$-vertex graph G and an integer k such that ϕ is satisfiable if and only if G has an independent set of size k. Therefore, assuming ETH, Independent Set cannot be solved in time $2^{o(n)}$ on n-vertex graphs, as this algorithm together with the reduction from 3SAT would give a $2^{o(m)}$ time algorithm for m-clause 3SAT. If we look at the literature on NP-hardness proofs, then we can see that many other hardness proofs have this property. From these hardness proofs, we can obtain results such as the following:

Corollary 2.2. *Assuming ETH, there is no* $2^{o(n)}$ *time algorithm for* Independent Set*,* Clique*,* Dominating Set*,* Hamiltonian Path *on* n*-vertex graphs.*

This means that every algorithm for these problems has to run in time exponential in the number of vertices or, in other words, there are no subexponential FPT algorithms parameterized by the number of the vertices. A colloquial term for algorithms that solve the problem in exponential time in the number of

vertices, possibly in a smart way, is "exact exponential-time algorithms" [52]; Corollary 2.2 can be interpreted as a lower bound on exact algorithms. However, as the number of vertices is an upper bound on the size of the solution, it also follows that there are no subexponential FPT algorithms parameterized by the size of the solution:

Corollary 2.3. *Assuming ETH, there is no $2^{o(k)} \cdot n^{O(1)}$ time algorithm for* Independent Set, Clique, Dominating Set, *and* k-Path, *where k is the size of the solution to be found.*

There are FPT problems for which there are subexponential-time parameterized algorithms. This is very common for planar problems: all the problems in Corollary 2.3 are known to be solvable in time $2^{O(\sqrt{k})} \cdot n^{O(1)}$ on planar graphs.[1] There are two main approaches for obtaining running time of this form on planar graphs: using planar separator results [3] and bidimensionality theory [36]. On the complexity side, if we look at the proofs showing the NP-hardness of these problems in planar graphs, then all of them involve "crossover gadgets" to deal with planarity. These gadgets induce a blowup in the size of the constructed instance: it is no longer linear in the number of clauses, but quadratic. Therefore, we get weaker lower bounds: we can only rule out the existence of algorithms with running time $2^{O(\sqrt{k})} \cdot n^{O(1)}$.

Corollary 2.4. *Assuming ETH, there is no $2^{o(\sqrt{k})} \cdot n^{O(1)}$ time algorithm for* Independent Set, Dominating Set, *and* k-Path *on planar graphs, where k is the size of the solution to be found.*

Note that these lower bounds match the known $2^{O(\sqrt{k})} \cdot n^{O(1)}$ time algorithms (up to the constant hidden by the big-O notation). These matching bounds can be considered a major success of the optimality program so far. Initially looking at planar problems, it is not obvious why square root is the function that should appear in the running time, but we have learned that this is an inherent feature of planarity and now we have a good understanding of both the upper bounds and the lower bounds.

A planar problem which is not fully understood yet is Subgraph Isomorphism: given graphs H and G, does G has a subgraph isomorphic to H? On planar graphs, the problem is known to be solvable in time $2^{O(k)} \cdot n^{O(1)}$ [39], where k is the number of vertices of H (improving an earlier $k^{O(k)} \cdot n^{O(1)}$ algorithm [45]). Could it be that square root appears in this problem as well and the running time can be improved further to $2^{O(\sqrt{k})} \cdot n^{O(1)}$? There is no known complexity result ruling out this possibility. Furthermore, significantly new techniques would be required to rule out the existence of a $2^{o(k)} \cdot n^{O(1)}$ algorithm for the problem: as the problem is planar, typical reductions need to introduce crossover gadgets, which would create a blowup in the size of the instance.

Subexponential-time FPT results are fairly standard for planar problems. It is much more surprising if a problem on general graphs admits a subexponential-time algorithm. Very recently, this turned out to be the case for the Chordal

[1] Actually, Clique can be solved in polynomial time on planar graphs.

Completion problem (given a graph G and an integer k, decide if G can be made chordal by adding at most k edges). Various $2^{O(k)} \cdot n^{O(1)}$ time algorithms are known for the problem [17, 65, 15]. Fomin and Villanger [54] gave a significant improvement by presenting a $2^{O(\sqrt{k}\log k)} \cdot n^{O(1)}$ time algorithm. It is an interesting question whether this running time can be further improved. As observed in [54], the NP-hardness proofs imply that, assuming ETH, there is no $2^{o(k^{1/6})} \cdot n^{O(1)}$ time algorithm. Therefore, currently there is a large gap between the best upper and lower bounds.

Obtaining lower bounds of the form $2^{o(k)}n^{O(1)}$ or $2^{o(\sqrt{k})}n^{O(1)}$ on parameterized problems generally follows from the known NP-hardness reductions. However, there are some parameterized problems where $f(k)$ is "slightly superexponential" in the best known running time: $f(k)$ is of the form $k^{O(k)} = 2^{O(k \log k)}$. Algorithms with this running time naturally occur when a search tree of height at most k and branching factor at most k is explored, or when all possible permutations, partitions, or matchings of a k element set are enumerated. In many cases, the $f(k)$ running time was later improved to $2^{O(k)}$, often with significant extra work or with the introduction of a new technique. We have seen an example of this with the Subgraph Isomorphism problem on planar graphs. Another example: Monien [85] in 1985 gave a $k! \cdot n^{O(1)}$ time algorithm for finding a cycle of length k in a graph on n vertices. Alon, Yuster, and Zwick [6] introduced the color coding technique in 1995 and used it to show that a cycle of length k can be found in time $2^{O(k)} \cdot n^{O(1)}$. A very recent example is the case of the Hamiltonian Cycle problem parameterized by treewidth. A $w^{O(w)} \cdot n^{O(1)}$ time algorithm for graphs of treewidth w follows from standard dynamic programming techniques (see e.g., [50]). Very recently, Cygan et al. [30] introduced an elegant new technique called cut and count, and used it to design a (randomized) algorithm that, given a tree decomposition of width w, solves the problem in time $4^w \cdot n^{O(1)}$.

However, there are still a number of problems where the best running time seems to be "stuck" at $2^{O(k \log k)} \cdot n^{O(1)}$. Recently, for some of these problems matching lower bounds excluding running times of the form $2^{o(k \log k)} \cdot n^{O(1)}$ were obtained under ETH [75] (see also [30] for further examples), showing the optimality of these algorithms.

- The pattern matching problem Closest String (given k strings over an alphabet Σ and an integer d, decide if there is a string whose Hamming-distance is at most d from each of the k strings) is known to be solvable in time $2^{O(d \log d)} \cdot n^{O(1)}$ [57] or $2^{O(d \log |\Sigma|)} \cdot n^{O(1)}$ [77]. Assuming ETH, there is no $2^{o(d \log d)} \cdot n^{O(1)}$ and $2^{o(d \log |\Sigma|)} \cdot n^{O(1)}$ time algorithms [75].
- The graph embedding problem Distortion (decide whether a graph G has a metric embedding into the integers with distortion at most d) can be solved in time $2^{O(d \log d)} \cdot n^{O(1)}$ [47]. Assuming ETH, there is no $2^{o(d \log d)} \cdot n^{O(1)}$ time algorithm [75].
- The Disjoint Paths problem can be solved in time in time $2^{O(w \log w)} \cdot n^{O(1)}$ on graphs of treewidth at most w [95]. Assuming ETH, there is no $2^{o(w \log w)} \cdot n^{O(1)}$ time algorithm [75].

We expect that many further results of this form can be obtained by using the framework of [75]. Thus the existence of parameterized problems requiring "slightly superexponential" time $2^{O(k \log k)} \cdot |I|^{O(1)}$ is not a shortcoming of algorithm design or a pathological situation, but an unavoidable feature of the landscape of parameterized complexity.

The results discussed so far show the optimality of some $2^{O(k)} \cdot n^{O(1)}$, $2^{O(\sqrt{k})} \cdot n^{O(1)}$, and $2^{O(k \log k)} \cdot n^{O(1)}$ time algorithms. Are there natural problems for which the optimum running time is of some other form, say, $2^{O(k^2)} \cdot n^{O(1)}$ or $2^{2^{O(k)}} \cdot n^{O(1)}$? The curious problem Clique-or-Independent-Set (given a graph G and an integer k, is there a set of k vertices that induce a clique *or* an independent set?) can be solved in time $2^{O(k^2)} \cdot n^{O(1)}$ using a simple Ramsey argument ([69, 67]), thus it could be a candidate problem where this form of running time is optimal. Planar Deletion (delete k vertices to make the graph planar) could be a candidate for a natural problem where double-exponential dependence on k is necessary. The fixed-parameter tractability results for Planar Deletion [83, 66] depend on solving the problem on bounded-treewidth graphs, and it seems that the natural algorithm based on destroying all K_5 and $K_{3,3}$ subdivisions have double-exponential dependence on treewidth.

A more ambitious project is to understand the exact constants in the function $f(k)$ for the problem: for example, what is the smallest $c > 0$ such that there is a $c^k \cdot n^{O(1)}$ time algorithm for the problem? Let us note first that obtaining such results is very different and much more challenging than proving lower bounds of the form, say, $2^{o(k)} \cdot n^{O(1)}$. The problem is that determining the best possible c is machine-model dependent in the sense that it is not robust under polynomial-transformations of the running time. That is, a $4^k \cdot n^{O(1)}$ running time is just the square of $2^k \cdot n^{O(1)}$. ETH as formulated in Conjecture 2.1, however, is invariant under polynomial transformations of the running time: any polynomial of $2^{o(m)}$ is still $2^{o(m)}$. Therefore, it seems unlikely that such a coarse conjecture would give an easy way of proving the fine distinctions between running times $c^k \cdot n^{O(1)}$ for different values of c. A more suitable conjecture is the Strong Exponential Time Hypothesis (SETH); for the purposes of this paper, we can state it the following way:

Conjecture 2.5 (Strong Exponential Time Hypothesis [63, 18]) *There is no* $(2-\epsilon)^n \cdot m^{O(1)}$ *time algorithm for* n*-variable* m*-clause* SAT *for any* $\epsilon > 0$.

Note that here SAT is the satisfiability problem with unbounded clause size. For fixed clause size, there are better algorithms, see e.g., [60]. Lokshtanov et al. [74] used SETH to prove tight lower bounds on algorithms working on tree decompositions. Suppose that we want to solve a problem on a graph G and a tree decomposition of width w of G is given in the input. Assuming SETH, for every $\epsilon > 0$

- Independent Set cannot be solved in $(2-\epsilon)^w |V(G)|^{O(1)}$ time,
- Dominating Set cannot be solved in $(3-\epsilon)^w |V(G)|^{O(1)}$ time,
- Max Cut cannot be solved in $(2-\epsilon)^w |V(G)|^{O(1)}$ time,
- Odd Cycle Transversal cannot be solved in $(3-\epsilon)^w |V(G)|^{O(1)}$ time,

– For any $q \geq 3$, q-Coloring cannot be solved in $(q-\epsilon)^w|V(G)|^{O(1)}$ time,
– Partition Into Triangles cannot be solved in $(2-\epsilon)^w|V(G)|^{O(1)}$ time.

These lower bounds match the best known algorithms for the problem (up to the ϵ in the base of the exponent). Some further lower bounds of this form can be found in [30]. It seems to be a very different and significantly more challenging task to prove such tight results for problems parameterized by the size of the solution (instead of treewidth). The natural targets for such lower bounds are problems where the best known algorithms have running times of the form $c^k \cdot n^{O(1)}$ for some integer c. Cygan et al. [30] gave such (randomized) algorithms for a number of problems using the technique of cut and count.

The optimality results we have discussed so far make fixed-parameter tractability quantitative: we not only know now that the problem is FPT, but we also know what the best $f(k)$ in the running time can be. Another aspect of the optimality program is to make W[1]-hardness results quantitative. That is, instead of just knowing that the problem is not FPT and therefore the parameter has to appear in the exponent of the running time, we would like to know how exactly the exponent should depend on the parameter. A W[1]-hardness result by itself does not rule out the possibility that the problem can be solved in, say, time $2^k \cdot n^{O(\log\log\log\log k)}$, which would be "morally equivalent" to fixed-parameter tractability.

The Exponential Time Hypothesis can be used to give a tight lower bound on the exponent of the running time. Chen et al. [22] showed that for the Clique problem the $n^{O(k)}$ brute force algorithm is already optimal in this respect:

Theorem 2.6. *Assuming ETH,* Clique *cannot be solved in time $f(k) \cdot n^{o(k)}$ for any computable function f.*

Using parameterized reductions, we can transfer the lower bound of Theorem 2.6 to other problems. The exact form of the lower bound depends on how the parameterized reduction changes the parameter. For the following problems, the reductions increase the parameter at most by a constant factor, thus we get a lower bound of the same form:

Theorem 2.7. *Assuming ETH,* Independent Set *and* Dominating Set *cannot be solved in time $f(k) \cdot n^{o(k)}$ for any computable function f.*

On the other hand, if the reduction increases the parameter by more than a constant factor, then the lower bound gets weaker. For example, a reduction from Clique (on general graphs) to Dominating Set on unit disk graphs was presented in [79], which increases the parameter from k to $O(k^2)$. Therefore, we have the following lower bound:

Theorem 2.8. *Assuming ETH,* Dominating Set *on unit disk graphs cannot be solved in time $f(k) \cdot n^{o(\sqrt{k})}$ for any computable function f.*

As Dominating Set on unit disk graphs can be solved in time $n^{O(\sqrt{k})}$ [4], Theorem 2.8 is tight. Thus, similarly to many planar problems, the appearance

of the square root in the running time can be an inherent feature of geometric problems.

Most W[1]-hardness results in the literature are from Clique (or Independent Set, which is the same). Therefore, by analyzing how the parameter changes in the reduction, we can extract lower bounds similar to the ones above by transfering Theorem 2.6 to the problem at hand. One should examine it separately for each problem whether the lower bound obtained this way is tight or not. Many of the more involved reductions from Clique use edge selection gadgets (see e.g., [48, 51, 79]). As a clique of size k has $\Theta(k^2)$ edges, this means that the reduction typically increases the parameter to $\Theta(k^2)$ at least and, similarly to Theorem 2.8, what we can conclude is that there is no $f(k)n^{o(\sqrt{k})}$ time algorithm for the target problem (unless ETH fails). If we want to obtain stronger bounds on the exponent, then we have to avoid the quadratic blow up of the parameter and do the reduction from a different problem. Many of the reductions from Clique can be turned into a reduction from the more general Subgraph Isomorphism (Given two graphs H and G, decide if H is a subgraph of G). In a reduction from Subgraph Isomorphism, we need $|E(H)|$ edge selection gadgets, which usually implies that the new parameter is $\Theta(|E(H)|)$. Thus the following lower bound on Subgraph Isomorphism, parameterized by the number of edges in H, could be used to obtain tighter lower bounds compared to those coming from the reduction from Clique.

Theorem 2.9 ([81]). *If* Subgraph Isomorphism *can be solved in time $f(k)n^{o(k/\log k)}$, where f is an arbitrary function and k is the number of edges of the smaller graph H, then ETH fails.*

We remark that it is an interesting open question if the factor $\log k$ in the exponent can be removed, making this result tight (and also making the results following from Theorem 2.9 tighter).

Closest Substring (a generalization of Closest String) is an extreme example where reductions increase the parameter exponentially or even double exponentially, and therefore we obtain very weak lower bounds. In this problem, the input consists of strings $s_1, \ldots, s_t$ over an alphabet Σ and integers L and d. The task is to find a string s of length L such that every s_i has a consecutive substring s'_i of length L with Hamming-distance at most d from s.

Let us restrict our attention to the case where the alphabet is of constant size, say binary. Marx [80] gave a reduction from Clique to Closest Substring where $d = 2^{O(k)}$ and $t = 2^{2^{O(k)}}$ in the constructed instance (k is the size of the clique we are looking for in the original instance). Therefore, we get weak lower bounds with only $o(\log d)$ and $o(\log\log k)$ in the exponent. Surprisingly, these lower bounds are actually tight, as there are algorithms matching these bounds.

Theorem 2.10 ([80]). Closest Substring *over an alphabet of constant size can be solved in time $f(d)n^{O(\log d)}$ or in $f(d,k)n^{O(\log\log k)}$. Furthermore, assuming ETH, there are no algorithms for the problem with running time $f(k,d)n^{o(\log d)}$ of $f(k,d)n^{o(\log\log k)}$.*

While the results in Theorems 2.6 and 2.8 are asymptotically tight, they do not tell us the exact form of the exponent, that is, we do not know what the smallest c is such that the problems can be solved in time n^{ck}. However, assuming SETH, stronger bounds of this form can be obtained. Specifically, Pătraşcu and Williams [88] obtained the following bound for Dominating Set under SETH.

Theorem 2.11 ([88]). *Assuming SETH, there is no $O(n^{k-\epsilon})$ time algorithm for* Dominating Set *for any $\epsilon > 0$ and $k \geq 2$.*

3 Kernelization from the viewpoint of optimality

Kernelization is one of the most basic and most practical algorithmic techniques in parameterized complexity. Recall that a *kernelization* for a parameterized problem P is a polynomial-time algorithm that, given an instance I of P with parameter k, produces another instance I' of P with parameter k' such that (1) I is a yes-instance if and only if I' is a yes-instance, (2) the size of I' is at most $f(k)$ for some computable function f, and (3) k' is at most $f(k)$. Intuitively, one can think of a kernelization as a fast preprocessing algorithm producing a small "hard core" of the problem that needs to be solved. We say that a kernelization is an *$f(k)$-kernel* if the size of I' is at most $f(k)$. For graph problems, we also use the term *$f(k)$-vertex-kernel* to indicate that I' has at most $f(k)$ vertices.

If a parameterized problem admits a kernel, then this immediately implies that the problem is FPT. We can use the kernelization algorithm to produce an equivalent instance I' of size at most $f(k)$, and then we can use any brute force algorithm to solve I' in time that can be bounded by a function of k (assuming the problem is decidable). More surprisingly, a folklore result shows that the reverse direction is also true:

Theorem 3.1. *A decidable parameterized problem has a kernel if and only if it is* FPT.

Proof. We have seen the forward direction above. For the reverse direction, suppose that a parameterized problem can be solved in time $f(k) \cdot n^c$ for some computable function $f(k)$ and constant c. Given an instance I of the problem, let us simulate this algorithm for n^{c+1} steps. If the algorithm terminates during this simulation, then we can produce a kernel by outputing a trivial yes- or a trivial no-instance. If the $f(k) \cdot n^c$ time algorithm does not terminate in n^{c+1} steps, then $n < f(k)$. This means that I itself is a kernel with size at most $f(k)$. □

What does Theorem 3.1 tell us? It suggests that every FPT result can be explained as a kernelization together with an exact algorithm. Thus the study of fixed-parameter tractability can be reduced to the study of kernelization algorithms and exact exponential-time algorithms (or in other words, parameterization by the size of the instance). Given the breadth of techniques in parameterized complexity that does not seem to have anything to do with these two

concepts (e.g., color coding, iterative compression, and algebraic techniques), this is a somewhat disheartening and suspicious claim.

Let us revisit this claim from the viewpoint of the optimality program. It is true that every fixed-parameter tractability result can be obtained as a combination of kernelization and exact algorithms, but is it the *right way* of solving the problem? That is, can we get the best possible (or at least a reasonable good) $f(k)$ in the running time this way? For some problems this seems to be the case. For example, a classical result of Nemhauser and Trotter [86] shows that Vertex Cover admits a $2k$-vertex-kernel and can be solved trivially in time $2^{O(n)}$ on n-vertex graphs. This results in a $2^{O(k)} \cdot n^{O(1)}$ time algorithm, which is the optimal form of the running time by Corollary 2.3. For Dominating Set on planar graphs, several $O(k)$-vertex-kernels are known and the problem can be solved in time $2^{O(\sqrt{n})}$ on n-vertex planar graphs either by treewidth techniques or by using planar separator theorems. This combination gives us $2^{O(\sqrt{k})} \cdot n^{O(1)}$ time algorithms, which matches the lower bound of Corollary 2.4.

For other problems, however, we cannot reach the best possible running time using this combination. In order to show this, we need a way of proving lower bounds on the size of kernels that can be achieved. Bodlaender et al. [13], using the work of Fortnow and Santhanam [55], developed a technique for showing (modulo a complexity assumption) that certain problems do not admit kernels of polynomial sizes. This result started a whole new line of research and the technique has been subsequently used in several papers to prove similar lower bounds. We state only one such result here:

Theorem 3.2 ([13]). *Assuming* coNP $\not\subseteq$ NP/poly*, there is no* $k^{O(1)}$*-kernel for* k-Path*.*

The assumption coNP $\not\subseteq$ NP/poly is a fairly standard complexity assumption, for example, if it is false, then the polynomial hierarchy collapses [98].

The k-Path problem is known to be solvable in time $2^{O(k)} \cdot n^{O(1)}$ by various techniques [6, 10, 97]. Can we match this running time by a combination of kernelization and exact algorithms? Clearly, we can solve k-Path by a brute force exact algorithm in time $n^{O(k)}$. By Theorem 3.2, we cannot produce a kernel with k^c vertices for any constant c, thus this combination cannot even guarantee a running time of $k^{ck} \cdot n^{O(1)}$ for any constant c. In other words, even though Theorem 3.1 shows that k-Path has a kernelization algorithm and therefore in principle we could obtain FPT algorithms for the problem by kernelization followed by an exact algorithm, this is not the right combination of techniques to solve the problem, as it cannot reach the best possible running time $2^{O(k)} \cdot n^{O(1)}$.

We can argue similarly for other problems where the existence of a polynomial kernel can be ruled out by a result analogous to Theorem 3.2. But what about problems for which polynomial kernels do exist? Very recently, some results appeared that give tight lower bounds for problems admitting polynomial kernels. Recall that given a collection of sets of size d of a set of elements and an integer k, the d-Set Cover asks for a set of at most k elements that intersects every set in the input, while the d-Set Packing problem asks for k pairwise-disjoint sets from the collection. For both problems, algorithms based

on the Sunflower Lemma give kernels containing at most k^d sets (see e.g., [34]). The following results show that this is essentially best possible:

Theorem 3.3 ([35]). *Assuming* coNP $\not\subseteq$ NP/poly*, there is no* $O(k^{d-\epsilon})$*-kernel for* d-Set Cover *for any* $d \geq 3$ *and* $\epsilon > 0$*.*

Theorem 3.4 ([34]). *Assuming* coNP $\not\subseteq$ NP/poly*, there is no* $O(k^{d-\epsilon})$*-kernel for* d-Set Packing *for any* $d \geq 3$ *and* $\epsilon > 0$*.*

The d-Set Cover problem can be solved in time $d^k \cdot n^{O(1)}$ by simple branching and d-Set Packing can be solved in time $2^{O(dk)} \cdot n^{O(1)}$ for example by color coding. Can we match these running times by a combination of kernelization and exact algorithms? It is not clear at this point. Both problems can be solved in time $2^{O(n)}$ if n is the number of elements (this is obvious for d-Set Cover, as we can try every subset of the elements; for d-Set Packing, this follows from standard dynamic programming techniques). Therefore, the questions is whether kernels with $O(k)$ *elements* exist for these problems. Theorems 3.3–3.4 do not rule out this possibility, as they give a lower bound on the number *sets* only. Note that the current best upper bounds on the number of elements in the kernel are far from being $O(k)$ [1, 2]. It is a very interesting and challenging question for further research to understand what the best possible bound is in terms of the number of elements. From the viewpoint of the optimality program, one needs to answer this question in order to evaluate whether kernelization is the right way of solving these problems, or other techniques such as branching and color coding are inherently necessary to achieve the best possible running time.

Finally, for problems that admit linear (vertex-)kernels, one would like to know the best possible constant factor. For example, is there a $(2-\epsilon)k$-vertex-kernel for Vertex Cover? There is a simple 2-approximation for this problem and there is no $(2-\epsilon)$-approximation under the Unique Games Conjecture [68]. It seems to be too much of a coincidence that the same number appears both in the best kernel and the best approximation. This could be the sign that there are some deep connections that we are unaware of at the moment.

Chen et al. [20] proposed an elegant argument for proving lower bounds on kernel size. The *parametric dual* of a parameterized problem with respect to a size function s is the same problem, but now we consider $s - k$ the parameter instead of k. For example, the parametric dual of Vertex Cover with respect to the number of vertices is Independent Set (since there is a vertex cover of size k in an n-vertex graph if and only if there is an independent set of size $n-k$). Chen et al. [20] showed that if a parameterized problem and its dual both admit small kernels of linear size, then one can solve the instance by repeated applications of the two kernelization algorithms. This technique is very useful for planar or bounded-degree problems, as for these classes it is fairly natural that both the problem and its parametric dual have linear kernels. Let us state as an example a few lower bounds that follow from this technique:

Theorem 3.5. *[20] Assuming* P $\neq$ NP*, for any* $\epsilon > 0$

– Vertex Cover *on planar graphs does not have a* $(\frac{4}{3}-\epsilon)k$*-vertex-kernel.*
– Vertex Cover *on planar triangle-free graphs does not have a* $(\frac{3}{2}-\epsilon)k$*-vertex-kernel.*
– Independent Set *on planar graphs does not have a* $(2-\epsilon)k$*-vertex-kernel.*
– Dominating Set *on planar graphs does not have a* $(2-\epsilon)k$*-vertex-kernel.*

Note, however, that these result do not give lower bounds on kernelization for general graphs: a kernelization algorithm for general graphs can transform a planar instance into a nonplanar one, hence it is not necessarily a correct kernelization algorithm for the planar problem as well.

We conclude this section by pointing out two technical issues that have arisen in the study of kernelization. In the definition of kernelization, we want to bound the size of the constructed instance. However, we might want to bound some other measure instead, for example, the number of vertices in the graph. From the practical point of view, for most graph-theoretical problems the time required for the exact solution of the kernel is mainly influenced by the number of vertices, thus it makes sense to focus on reducing the number of vertices. On the other hand, bounding the size of the instance seems to be mathematically more robust question, for example, the techniques of [13, 35] give primarily lower bounds on the size of the instance. Both kind of bounds are worth studying, but we have to make a clear distinction between the two types of results and realize the different consequences.

Another technical issue is the bound on the parameter in the kernel. Originally, Downey and Fellows [44] required that the parameter of the kernel is at most the parameter of the original instance. This makes sense: as we imagine that the parameter measures the hardness of the instance, we do not want the preprocessing to increase it. Later, e.g., in [14, 13] a more liberal definition is given, where we only require that the new parameter is bounded by a function of the old parameter (we used this definition in the beginning of the section). An advantage of this definition is that it is robust with respect to polynomial transformations of the kernel. For example, we can create a polynomial-size kernel that is an instance of some other problem (this is sometimes called a *bikernel*) and then use a polynomial-time reduction to transform it into an instance of the original problem. This results in a polynomial-size kernel, but the parameter can increase in the reduction. Allowing such arguments in proving the existence of polynomial-size kernels makes the theory more robust and mathematically more natural, although it weakens the connection with practical preprocessing. One has to be aware of this difference and interpret the results accordingly.

4 Branching algorithms

Besides kernelization, the technique of "bounded-depth search trees" is perhaps the most basic method for showing that a problem is fixed-parameter tractable. Let us recall how this technique works in the case of Vertex Cover. Let G be a graph where a vertex cover of size k has to be found. Let $e = uv$ be an arbitrary edge of G. Clearly, every vertex cover contains either u or v (or both).

Therefore, we branch into two directions. In the first branch, we assume that u is in the vertex cover, hence we try to find recursively a vertex cover of size $k-1$ in $G \setminus u$. In the second branch, we assume that v is in the vertex cover and try to find recursively a vertex cover of size $k-1$ in $G \setminus v$. Clearly, if there is a solution, at least one of the two branches finds a solution. We repeat this branching step until there is no edge in the graph or k becomes 0. Running this recursive process creates a search tree where each node has at most two children. The crucial property to observe is that the height of the search tree is at most k: the parameter strictly decreases in each step. Therefore, the search tree has at most 2^k leaves and hence $O(2^k)$ nodes. Each recursion step can be done in polynomial time, hence it follows that the total running time is $2^k \cdot n^{O(1)}$. The d-Set Cover problem is a generalization of Vertex Cover: given sets of size d, we have to find k elements that hit every set. In a similar way, one can obtain a $d^k \cdot n^{O(1)}$ algorithm for d-Set Cover by selecting a set and branching on which element of the set is included in the solution.

In summary, the main idea of the bounded-depth search tree technique is to reduce the instance into a bounded number of instances with strictly smaller parameter values. If the reduction creates at most c instances, then the running time is $c^k \cdot n^{O(1)}$. In some cases, the number of directions we branch into depends also on the parameter. For example, if we create at most k instances in each step, then we can bound the running time by $k^k \cdot n^{O(1)}$. The $d^d \cdot n^{O(1)}$ time algorithm for Closest String [57] mentioned in Section 2 is an example of such a branching algorithm.

Seeing how fruitful the systematic analysis of kernelization turned out to be, one wonders why there haven't been any systematic analysis of the applicability of branching algorithms. The purpose of this section is to propose a framework in which this question can be studied. What we have learned in the study of kernelization is that one should pay attention to optimality: the question is not whether branching algorithms *can* be used to solve a problem, but whether it is the *right way* of solving the problem. Therefore, here we stick to the study of $c^k \cdot n^{O(1)}$ time algorithms that branch into a constant number of directions. In particular, we are not interested in the question whether there is a $k^k \cdot n^{O(1)}$ time branching algorithm for a problem that can be solved in time $c^k \cdot n^{O(1)}$ by other techniques (because such a branching algorithm would be far from being the optimal way of solving the problem).

Let us formalize first the notion of a branching rule.

Definition 4.1. *Let (I, k) be an instance of a parameterized problem with $k > 1$. A* c-way branching rule *for some constant c is a polynomial-time algorithm that, given instance I, produces instances (I_1, k_1), ..., (I_c, k_c) such that*

1. *$|I_i| \leq |I|$ for every $1 \leq i \leq c$,*
2. *$k_i < k$ for every $1 \leq i \leq c$,*
3. *(I, k) is a yes-instance if and only if (I_i, k_i) is a yes-instance for at least one $1 \leq i \leq c$.*

It is easy to see that if a parameterized problem has a c-way branching rule, then we can solve the problem in time $c^k \cdot n^{O(1)}$ (assuming the problem is polynomial-

time solvable for $k = 1$, which is the case for the problems we are interested in). The algorithm described at the beginning of the section shows that Vertex Cover has a 2-way branching rule. Thus it seems that we have a simple framework for formally studying which problems can be solved by the technique of bounded-depth search trees.

Unfortunately, there are parameterized problems that do not have branching rules in the sense of Definition 4.1 for pathological reasons. For example, consider the (artificial) problem Vertex Cover↑ defined as follows: given a graph G and an integer k, the task is to decide of G has a vertex cover of size k and, additionally, if $k = 2^i$ for some integer i (i.e., k is a power of 2). Clearly, this problem is not more complicated than Vertex Cover: all we need is the trivial extra check whether k is a power of 2. Still, this problem has no branching rule:

Proposition 4.2. *Assuming* P ≠ NP*,* Vertex Cover↑ *does not have a branching rule.*

Proof. A simple padding argument shows that Vertex Cover↑ is NP-hard. Suppose that $\mathcal{A}$ is a branching algorithm for Vertex Cover↑ that produces a constant number c of instances. We can assume that for every instance (I_i, k_i) created by $\mathcal{A}$, parameter k_i is a power of 2, since otherwise (I_i, k_i) is trivially a no-instance. Furthermore, we can assume that we run $\mathcal{A}$ only on instances whose parameter is a power of 2. Therefore, if the parameter is 2^i, algorithm $\mathcal{A}$ creates c instances with parameter at most 2^{i-1}. This means that the height of the search tree is at most $\log_2 k$ and therefore the size of the search tree is $O(c^{\log_2 k}) = O(k^{\log_2 c})$, which is polynomial in k (as c is a fixed constant). Thus we can solve Vertex Cover↑ in polynomial time, implying P = NP. □

To avoid situations like Proposition 4.2, we have to allow that a branching algorithm solves a modified version of the problem (e.g., Vertex Cover instead of Vertex Cover↑). We express this by saying that we are interested in problems that can be reduced to a problem that has a branching rule. The right notion of reduction for this purpose is a restriction of parameterized reduction that runs in polynomial time and the parameter can be increased only by at most a constant factor:

Definition 4.3. *A* linear-parameter polynomial-time parameterized transformation (LPPT) *from a parameterized problem P_1 to a parameterized problem P_2 is a polynomial-time algorithm that, given an instance (I_1, k_1) of P_1, creates an instance (I_2, k_2) of P_2 such that*

1. *(I_1, k_1) is a yes-instance of P_1 if and only if (I_2, k_2) is a yes-instance of P_2, and*
2. *$k_2 \leq c \cdot k_1$ for some constant c.*

Now we can define the class BFPT(where B stands for "branching"), which formalizes the notion of branching:

Definition 4.4. *The class* BFPT *contains a parameterized problem P_1 if there is a parameterized problem P_2 that has a branching rule and there is an LPPT from P_1 to P_2.*

Let us observe that Vertex Cover↑ is in BFPT as expected: there is a trivial LPPT from this problem to Vertex Cover.

Before discussing further examples of problems in BFPT, let us show a simple equivalent characterization of BFPT with linear-size witnesses. Recall that a language P is in NP if there is a polynomial-time decidable language P' and a polynomial p such that $x \in P$ if and only if there is a string w (the *witness*) of length at most $p(|x|)$ such that $(x, w) \in P'$. Informally, we can say that w is a polynomial-size witness for x that can be verified in polynomial. The following lemma shows that BFPT contains those NP languages where there is a witness whose length is linear in the parameter.

Lemma 4.5. *A parameterized problem is in* BFPT *if and only if there is a polynomial-time decidable language P' and a constant c such that $(x, k) \in P$ if and only if there is a string w of length at most $c|k|$ such that $(x, k, w) \in P'$.*

Proof. For the forward direction, suppose that parameterized problem P can be LPPT-reduced to a parameterized problem Q that has a c-way branching algorithm $\mathcal{A}$. Given an instance (I, k) of P, let (I', k') be the instance of Q created by the LPPT reduction. If $(I', k') \in Q$, then one of the branches of $\mathcal{A}$ is successful, i.e., produced a yes-instance with parameter $k = 1$. As $\mathcal{A}$ branches into c directions, we can describe with $\lceil \log_2 c \rceil \cdot k' = O(k)$ bits a successful branch. This description is a good witness for (I, k): one can verify it in polynomial-time by computing the instance (I', k') given by the LPPT-reduction and then verifying that this branch of the search tree of $\mathcal{A}$ is indeed successful.

For the reverse direction, let us define the language P'' such that $(x, k, w, \ell) \in P''$ if there is a string q of length at most ℓ such that $(x, k, wq) \in P'$. In other words, $(x, k, w, \ell) \in P''$ means that w can be extended with at most ℓ bits to a witness of (x, k). The problem P'' parameterized by ℓ has a branching rule: we try to append a 0 or a 1 to w. Formally, $(x, k, w, \ell) \in P''$ if and only if either $(x, k, w) \in P'$ (which can be checked in polynomial time), or $(x, k, w0, \ell - 1) \in P''$, or $(x, k, w1, \ell - 1) \in P''$. By assumption, $(x, k) \in P$ if and only if there is a string w of length at most $c|k|$ such that $(x, k, w) \in P'$, or equivalently, $(x, k, \epsilon, c|k|) \in P''$ (where ϵ is the empty word). This gives an LPPT-reduction from P to P'', a problem that has a branching algorithm. □

Lemma 4.5 gives a more convenient way of showing that a problem is in BFPT. There are many examples of branching algorithms where the parameter does not necessarily decrease after each branching step, but we can show that some other measure strictly decreases. In such a case, it would be awkward to use directly the definition of BFPT, since we need to define an artificial problem where the parameter is the measure bounding the height of the search tree. On the other hand, with Lemma 4.5 all we need to do is to observe that we branch into a constant number of directions in each step and the height of the search tree is bounded by a linear function of the parameter. Therefore, a string describing the successfully branch is a correct witness whose length is linear in the parameter.

As an example, let us use Lemma 4.5 to show that Node Multiway Cut (Given a graph G, a set of terminals $T \subseteq V(G)$, and an integer k, the task is to find a set S of at most k vertices that separates the terminals, that is, every component of $G \setminus T$ contains at most one vertex of T) is in BFPT. This problem is known to be FPT [78, 24, 31, 58]. Observation of, say, the $4^k \cdot n^{O(1)}$ algorithm of Chen et al. [24] shows that the search tree in the proof has height at most $2k$ and branching factor 2, thus there is a witness of $2k$ bits.

Proposition 4.6. Node Multiway Cut *is in* BFPT.

A standard technique in the design of parameterized algorithms is to solve the *compression* problem first. For example, let us consider the Feedback Vertex Set problem (given a graph G and an integer k, the task is to find a feedback vertex set of size k, that is, a set S of at most k vertices such that $G \setminus S$ is a forest). A randomized $4^k \cdot n^{O(1)}$ time algorithm was given in [8] and deterministic $2^{O(k \log k)} \cdot n^{O(1)}$ time algorithms were given already in [42, 11]. However, deterministic $c^k \cdot n^{O(1)}$ time algorithms appeared only much later and they all use the technique of compression [33, 59, 21, 30].

In the compression version of Feedback Vertex Set, the input contains additionally a feedback vertex set S_0 of size $k+1$. Intuitively, we have to "compress" a solution of size $k+1$ into a solution of size k. The compression problem can be easier than the original problem, as the initial solution S_0 can give us useful structural information about the graph. More generally, instead of starting with a solution having the specific size $k+1$, we can formulate the compression problem as starting with a solution of an arbitrary size $\ell > k$, and we parameterize by the problem by ℓ, the size of the initial solution.

There are two ways of using the compression algorithm to solve the original problem. The first method is to use the elegant technique of iterative compression, introduced by Reed et al. [92]. For a detailed explanation of this technique, see for example the survey [61]. The second method is to use a polynomial-time approximation algorithm to obtain a solution of size $f(k)$ and then use the compression algorithm to compress the initial solution of size $f(k)$ to a solution of size k (if such a solution exists). Let us observe that if we start with a constant-factor approximation and the compression is performed by a branching algorithm, then this combination yields a branching algorithm for the original problem. Therefore, we can state the following (somewhat informal) observation:

Proposition 4.7. *If a parameterized problem P has a polynomial-time constant-factor approximation and the compression version of P parameterized by the size of the initial solution is in* BFPT*, then P is in* BFPT.

Feedback Vertex Set has a 2-approximation and inspection of the proof, e.g., in [21] shows that the compression problem is in BFPT.

Proposition 4.8. Feedback Vertex Set *is in* BFPT.

There is an interesting connection between branching and kernelization. Suppose that a problem has a linear-vertex-kernel. Then the problem can be solved

by computing the kernel and doing a brute force search on it. If this brute force search can be done by branching, then this gives a branching algorithm for the problem. We can formalize this by the following statement:

Proposition 4.9. *If a parameterized problem P admits a linear-vertex-kernel and the version of the problem parameterized by the number of vertices is in* BFPT*, then P is in* BFPT*.*

For example, this gives an alternate way of seeing that Vertex Cover is in BFPT: it has a $2k$-vertex-kernel [86] and Vertex Cover parameterized by the number n of vertices can be trivially solved by branching, as it has a witness of n bits. Proposition 4.9 also applies to a wide range of planar problems. On planar graphs, many of the standard NP-hard problems become FPT and in fact admit linear-vertex-kernels; this follows for example from the powerful meta result of Bodlaender et al. [14]. For problems where the solution is a subset of vertices, it is usually trivial that the problem is FPT parameterized by the number of vertices, as a branching algorithm can enumerate all possible subsets. Therefore, we get for example the following results:

Proposition 4.10. Independent Set*,* Dominating Set*,* Connected Dominating Set*,* Connected Vertex Cover*,* Induced Matching *on planar graphs are in* BFPT*.*

However, let us note that Proposition 4.10 is somewhat unsatisfactory from the viewpoint of the optimality program. As these planar problems can be solved in time $c^{\sqrt{k}} \cdot n^{O(1)}$, it can be considered as irrelevant whether there are $c^k \cdot n^{O(1)}$ time branching algorithms for them.

Max Internal Spanning Tree (given a graph G and an integer k, the task is to find a spanning tree where at least k vertices are non-leaves, that is, have degree more than one) admits a $3k$-vertex-kernel, thus we might try to use Proposition 4.9 for this problem. However, it is not obvious if Max Internal Spanning Tree parameterized by the number of vertices has a branching algorithm. A branching algorithm can guess the internal vertices, but then one has to enforce somehow that the degrees of these vertices are more than one. It is therefore an interesting open question whether the problem, parameterized by k or by the number of vertices, is in BFPT.

The example of Max Internal Spanning Tree shows that the search for branching algorithms parameterized by the number n of vertices is also an interesting research question. This is particularly true for problems that can be solved in c^n time by dynamic programming techniques, for example, Hamiltonian Path, Chromatic Number, Partition into Triangles for graphs having n vertices, Set Packing over a universe of n elements, Hitting Set with n sets, etc. Paturi and Pudlák [89] raised a similar question: they ask if Hamiltonian Path has a polynomial-time randomized algorithm with success probability c^{-n} on n-vertex graphs. Note that if k-Path parameterized by the length k of the path is in BFPT, then this implies that k-Path parameterized by the number n of vertices is in BFPT, which further implies that Hamiltonian Path has the required randomized algorithm: we can replace branching

by random choices. This means that a negative answer to the question of Paturi and Pudlák would imply that k-Path is not in BFPT. Therefore, if one wants to show that there is no such randomized algorithm, probably it makes sense to concentrate on first showing that k-Path is not in BFPT, as this can be an easier question.

Branching algorithms are sometimes able to solve the more general counting version of the problem as well. This depends on the type of branching rule we use. If we know that every solution contains at least one element of a set S and we branch on the choice of exactly which subset of S is contained in the solution, then such a branching rule is usually good for counting: each solution remains a valid solution in exactly one of the branches, thus the number of solutions is exactly the sum of the number of solutions in all the branches. On the other hand, if we only know that whenever there is a solution, then there is also a solution containing an element of S and we branch on a subset of S, then this is typically not good for counting: we won't be able to count those solutions that are disjoint from S.

We could set up a framework for studying a stronger version of branching that is capable of solving counting problems. The main difference is that we want to require that the number of solutions is exactly the sum of the number of solutions in the different branches. We omit the details, as we do not have any interesting results at this point. The reason why we mention it, however, is the surprising fact that even though k-Path is FPT, the counting version is known to be #W[1]-hard [49]. Thus it is unlikely that it is fixed-parameter tractable and hence unlikely that it has a branching rule suitable for counting. An interesting possibility is that one might be able to transfer this negative result on counting branching rules to ordinary branching rules solving the decision problem. It could be that understanding counting problems is the key for understanding branching.

Let us briefly mention that there is another natural question about branching: for a problem that can be solved by branching, what is the best branching algorithm? One way to formulate this question is to ask what the smallest c is such that there is a c-way branching algorithm for the problem. However, this c is always an integer by definition, but most of the sophisticated branching rules are assymmetric (i.e., different branches reduce the parameter by different values) and the analyisis of such branching rules typically give a bound of c^k on the size of the search tree for some noninteger c. Therefore, probably it is a better and more relevant question to ask what the smallest c is such that the search tree is guaranteed to have size at most c^k. A different way to study the question is to find the smallest c such that there is a witness of size $c \cdot k$ for every instance. This question has been explored recenently for Sat by Dantsin and Hirsch [32].

The aim of this section was to point out that the theoretical study of which problems can be solved by branching algorithms has been neglected so far and it is possible to study this question in a rigorous framework. We have formulated meaningful and challenging questions for future work. Let us conclude this sec-

tion with a list of problems for which it would be interesting to decide if they are in BFPT:

- k-Path parameterized by the length k of the path or by the number n of vertices.
- Connected Vertex Cover parameterized by the size k of the solution.
- Steiner Tree parameterized by the number k of terminals.
- Hitting Set parameterized by the number of sets.
- Max Internal Spanning Tree parameterized by the number k of internal nodes in the tree.
- Chromatic Number parameterized by the number n of vertices.

5 Problems on directed graphs

Finding algorithms for problems on directed graphs is typically more challenging than solving their undirected counterparts. Many of the tools for undirected graphs become more complicated to use or even break down completely when we move to the domain of directed graphs. This jump in complexity has been observed also in the context of fixed-parameter tractability. For example, the Graph Minors Theory of Robertson and Seymour was the early inspiration for parameterized complexity and many powerful results followed from it almost immediately. However, there is no directed analog of the theory and hence directed graph problems have not received the same initial boost that undirected problems have.

Despite the inherent difficulty of directed problems, there has been some progress in this direction. The most celebrated such result is the fixed-parameter tractability of the Directed Feedback Vertex Set problem. The undirected Feedback Vertex Set problem was shown to be FPT already in 1992 [42] and subsequently several different algorithms have been found [11, 21, 33, 59]. The directed version (delete k vertices to make the graph acyclic) turned out to be much more challenging. It was not until 2008 that Chen et al. [25] proved the fixed-parameter tractability of Directed Feedback Vertex Set via a clever combination of iterative compression and solving directed cut problems. Looking at the proof, one can observe that the algorithm is fairly elementary and in particular it does not use any deep results of Graph Minors Theory. Perhaps the reason why the resolution of this problem took so long was that people looked for inspiration at the wrong place: it was expected that the solution is very complex and would somehow follow from a generalization of graph structure theory to directed graphs. For example, the fixed-parameter tractability of Feedback Vertex Set follows immediately from standard treewidth techniques [11]. Directed analogs of treewidth do exist [64, 7, 71, 62, 9], but apparently they do not provide any help for this problem. In fact, there is some formal evidence that no really useful directed width measure exists [56]. This means that when we are encountering directed problems, treewidth-based techniques, which are among the most useful theoretical tools in parameterized complexity, are missing from

our arsenal. Recently, however, there were some attempts to build a useful structure theory for directed graphs from a very different direction by generalizing the undirected notion of nowhere dense graphs [72].

One of the most useful applications of treewidth is bidimensionality theory [36]. This theory shows in a very easy way that subexponential-time parameterized algorithms exists for problems on planar and, more generally, on H-minor free graphs. It is a very interesting question whether subexponential-time algorithms follow with the same ease for directed planar problems. Dorn et al. [40] investigated this question, and gave $2^{O(\sqrt{k}\log k)} \cdot n^{O(1)}$ time algorithms for two problems on directed planar graphs, k-Leaf Out Branching and k-Internal Out Branching. For both problems, the key is to use treewidth techniques on the underlying undirected graph: using problem specific arguments and reductions, large grids can be excluded or standard layering techniques can be made to work. It is also observed in [40] that the directed version of k-Path for planar graphs can be solved in time $(1+\epsilon)^k \cdot n^{f(\epsilon)}$ for every $\epsilon > 0$. That is, the base of the exponent can be made arbitrary close to 1 at the cost of increasing the exponent of n. Note that obtaining this running time is a weaker claim than having a $2^{o(k)} \cdot n^{O(1)}$ time algorithm. Therefore, it remains a very interesting open question if Directed k-Path on planar graphs can be solved in time $2^{o(k)} \cdot n^{O(1)}$, or perhaps even in time $2^{O(\sqrt{k})} \cdot n^{O(1)}$.

Due to the strong modeling power of graphs, sometimes graph problems appear in disguise. For example, Almost 2SAT is the problem of deciding whether the given 2SAT formula has an assignment satisfying all but at most k clauses. While graphs do not appear explicitly in the problem definition, the first FPT algorithm by Razgon and O'Sullivan [91] considers a natural directed graph formed by the implications and solves the problem by finding separators in this directed graph. In general, whenever the problem involves chains of implications, one can expect directed graphs to make an appearance.

Hard problems on undirected graphs are often studied on particular subclasses of graphs: on planar graphs, interval graphs, bounded-treewidth graphs, etc. It is natural to do the same when a problem turns out to be hard on general directed graphs. First, one can restrict the problem to directed graphs whose underlying undirected graph has special structure. For example, it is an important general question whether the well-understood techniques for planar undirected graphs (bidimensionality, Baker's layering approach, etc.) work for problems on planar directed graphs. Furthermore, there are interesting classes of directed graphs that have no undirected counterparts. The class of directed acyclic graphs seems to be an obvious choice to try: these graphs have useful properties (for example, dynamic programming on a topological ordering can be a useful approach), but still many problems are nontrivial on this class. Another well-studied class is the class of tournaments: directed complete graphs, i.e., there is exactly one directed edge between any two distinct vertices (one can also consider the more general class of semicomplete directed graphs, where we allow bidirected edges as well). A classical result of Downey and Fellows [43] shows that Dominating Set is W[2]-complete for tournaments, that is, as hard

as on general (directed) graphs. This is somewhat surprising in light of the fact that Dominating Set on tournaments can be solved in time $n^{O(\log n)}$ and is not known to be NP-hard. Directed Feedback Arc Set was known to be FPT on tournaments [90] even before its fixed-parameter tractability in general graphs [25] was shown, but recently it turned out that on tournaments there are subexponential-time FPT algorithms for the problem: it can be solved in time $2^{O(\sqrt{k})} \cdot n^{O(1)}$ [5, 46]. Problems related to edge- or vertex-disjoint paths in tournaments have been studied recently [53, 27, 28], but some interesting questions are still open: for example, the k vertex-disjoint paths problem is polynomial-time solvable for every fixed value of k, but it is not known to be fixed-parameter tractable.

A family of problems that received particular attention is formed by problems related to cuts and separation. These problems usually have both edge deletion and vertex deletion versions; for simplicity we discuss only the vertex deletion version here. Given a graph G, a set $T \subseteq V(G)$ of terminals, and an integer k, the Node Multiway Cut problems asks for a set of at most k vertices whose deletion separates the terminals from each other. The problem is known to be FPT by various techniques [78, 24, 31, 58]. A more general problem is Multicut, where the input contains a set of terminal pairs (s_1, t_1), ..., (s_ℓ, t_ℓ) and we have to disconnect s_i from t_i for every i. The fixed-parameter tractability of Node Multiway Cut implies in an easy way that there is an $f(k, \ell) \cdot n^{O(1)}$ time algorithm for the problem, i.e., it is FPT with combined parameters k and ℓ. Very recently, it was shown that Multicut is FPT even if parameterized by k only [16, 82]. The directed versions of these problems are very different and less understood. Unlike the undirected problem, Directed Multicut is W[1]-hard parameterized by k [82]. However, the special case Directed Multiway Cut is FPT parameterized by k [26]. Somewhat surprisingly, the random sampling of important separators technique introduced in [82] for undirected Multicut works for the Directed Multiway Cut problem (but apparently it does not work for Directed Multicut, as it is W[1]-hard). It remains an interesting open question whether Directed Multicut is FPT with combined parameters k and ℓ. Note that the case $\ell = 2$ can be reduced to Directed Multiway Cut and hence is FPT by [26]. However, the case $\ell = 3$ is open. A possible motivation for the study of directed cut problems is that the solution of the Directed Feedback Vertex Set problem [25] relied in large part on the fixed-parameter tractability of a variant of Directed Multicut called Skew Multicut, where we have to disconnect s_i from every t_j with $i \geq j$. It could be fruitful to investigate what other disconnection requirement patterns make the problem tractable. Most of these questions are meaningful also on directed acyclic graphs. Very recently, it has been proved that Directed Multicut on directed acyclic graphs is fixed-parameter tractable with combined parameters k and ℓ, but W[1]-hard parameterized by k [70].

We conclude this section with two open questions that are easy to state but their solution would generalize and unify important known results. The first problem is Directed Eulerian Deletion (see [19, 29]): given a directed graph

G and an integer k, delete k vertices such that every strongly connected component induces an Eulerian directed graph, that is, a graph where the indegree equals the outdegree for every vertex. It is not difficult to see that Directed Feedback Vertex Set can be reduced to this problem, thus its solution should build on the arguments of [25]. The second problem is Directed Odd Cycle Transversal: given a graph G and an integer k, delete k vertices such that there is no directed odd cycle in the remaining graph. Easy reductions show that this problem is more general than Directed Feedback Vertex Set, Odd Cycle Transversal, and Directed Multiway Cut with $\ell = 2$. Thus a fixed-parameter tractability result for this problem would have to unify and generalize all the algorithmic ideas for these three problems.

References

1. Abu-Khzam, F.N.: An improved kernelization algorithm for r-set packing. Inf. Process. Lett. **110**(16), 621–624 (2010)
2. Abu-Khzam, F.N.: A kernelization algorithm for d-hitting set. J. Comput. Syst. Sci. **76**(7), 524–531 (2010)
3. Alber, J., Fernau, H., Niedermeier, R.: Parameterized complexity: exponential speed-up for planar graph problems. J. Algorithms **52**(1), 26–56 (2004)
4. Alber, J., Fiala, J.: Geometric separation and exact solutions for the parameterized independent set problem on disk graphs. J. Algorithms **52**(2), 134–151 (2004)
5. Alon, N., Lokshtanov, D., Saurabh, S.: Fast FAST. In: Proceedings of the 36th International Colloquium on Automata, Languages and Programming, ICALP 2009, Part I, *Lecture Notes in Computer Science*, vol. 5555, pp. 49–58. Springer Verlag (2009)
6. Alon, N., Yuster, R., Zwick, U.: Color-coding. J. ACM **42**(4), 844–856 (1995)
7. Barát, J.: Directed path-width and monotonicity in digraph searching. Graphs and Combinatorics **22**(2), 161–172 (2006)
8. Becker, A., Bar-Yehuda, R., Geiger, D.: Randomized algorithms for the loop cutset problem. J. Artif. Intell. Res. (JAIR) **12**, 219–234 (2000)
9. Berwanger, D., Dawar, A., Hunter, P., Kreutzer, S.: DAG-width and parity games. In: B. Durand, W. Thomas (eds.) Proceedings 23rd International Symposium on Theoretical Aspects of Computer Science, STACS 2006, *Lecture Notes in Computer Science*, vol. 3884, pp. 524–536. Springer Berlin / Heidelberg (2006)
10. Björklund, A.: Determinant sums for undirected hamiltonicity. In: Proceedings of the 51st Annual Symposium on Foundations of Computer Science, FOCS 2010, pp. 173–182 (2010)
11. Bodlaender, H.L.: On disjoint cycles. Int. J. Found. Comput. Sci. **5**(1), 59–68 (1994)
12. Bodlaender, H.L.: Kernelization: New upper and lower bound techniques. In: Proceedings of the 4th International Workshop on Parameterized and Exact Computation, IWPEC 2009, *Lecture Notes in Computer Science*, vol. 5917, pp. 17–37. Springer Verlag (2009)
13. Bodlaender, H.L., Downey, R.G., Fellows, M.R., Hermelin, D.: On problems without polynomial kernels. J. Comput. Syst. Sci. **75**(8), 423–434 (2009)
14. Bodlaender, H.L., Fomin, F.V., Lokshtanov, D., Penninkx, E., Saurabh, S., Thilikos, D.M.: (Meta) kernelization. In: Proceedings of the 50th Annual Symposium on Foundations of Computer Science, FOCS 2009, pp. 629–638 (2009)

15. Bodlaender, H.L., Heggernes, P., Villanger, Y.: Faster parameterized algorithms for minimum fill-in. Algorithmica **61**(4), 817–838 (2011)
16. Bousquet, N., Daligault, J., Thomassé, S.: Multicut is FPT. In: Proceedings of the 43rd Annual Symposium on Theory of Computing, STOC 2011, pp. 459–468 (2011)
17. Cai, L.: Fixed-parameter tractability of graph modification problems for hereditary properties. Inform. Process. Lett. **58**(4), 171–176 (1996)
18. Calabro, C., Impagliazzo, R., Paturi, R.: The complexity of satisfiability of small depth circuits. In: Proceedings of the 4th International Workshop on Parameterized and Exact Computation, IWPEC 2009, *Lecture Notes in Computer Science*, vol. 5917, pp. 75–85. Springer Verlag (2009)
19. Cechlárová, K., Schlotter, I.: Computing the deficiency of housing markets with duplicate houses. In: Proceedings of the 5th International Symposium on Parameterized and Exact Computation, IPEC 2010, *Lecture Notes in Computer Science*, vol. 6478, pp. 72–83. Springer Verlag (2010)
20. Chen, J., Fernau, H., Kanj, I.A., Xia, G.: Parametric duality and kernelization: Lower bounds and upper bounds on kernel size. SIAM J. Comput. **37**(4), 1077–1106 (2007)
21. Chen, J., Fomin, F.V., Liu, Y., Lu, S., Villanger, Y.: Improved algorithms for feedback vertex set problems. J. Comput. Syst. Sci. **74**(7), 1188–1198 (2008)
22. Chen, J., Huang, X., Kanj, I.A., Xia, G.: Linear FPT reductions and computational lower bounds. In: Proceedings of the 36th Annual Symposium on Theory of Computing, STOC 2004, pp. 212–221. ACM, New York (2004)
23. Chen, J., Kanj, I.A., Xia, G.: Improved upper bounds for vertex cover. Theor. Comput. Sci. **411**(40-42), 3736–3756 (2010)
24. Chen, J., Liu, Y., Lu, S.: An improved parameterized algorithm for the minimum node multiway cut problem. In: Proceedings of the 10th International Workshop on Algorithms and Data Structures, WADS 2007, *Lecture Notes in Computer Science*, vol. 4619, pp. 495–506. Springer Verlag (2007)
25. Chen, J., Liu, Y., Lu, S., O'Sullivan, B., Razgon, I.: A fixed-parameter algorithm for the directed feedback vertex set problem. J. ACM **55**(5) (2008)
26. Chitnis, R., Hajiaghayi, M., Marx, D.: Fixed-parameter tractability of directed multiway cut parameterized by the size of the cutset. In: Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, pp. 1713–1725 (2012)
27. Chudnovsky, M., Fradkin, A.O., Seymour, P.D.: Tournament immersion and cutwidth. J. Comb. Theory, Ser. B **102**(1), 93–101 (2012)
28. Chudnovsky, M., Scott, A., Seymour, P.: Vertex disjoint paths in tournaments. Manuscript.
29. Cygan, M., Marx, D., Pilipczuk, M., Pilipczuk, M., Schlotter, I.: Parameterized complexity of Eulerian deletion problems. In: Proceedings of the 37th International Workshop on Graph-Theoretic Concepts in Computer Science, WG 2011, *Lecture Notes in Computer Science*, vol. 6986, pp. 131–142. Springer Verlag (2011)
30. Cygan, M., Nederlof, J., Pilipczuk, M., Pilipczuk, M., van Rooij, J.M.M., Wojtaszczyk, J.O.: Solving connectivity problems parameterized by treewidth in single exponential time. In: Proceedings of the 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, pp. 150–159 (2011)
31. Cygan, M., Pilipczuk, M., Pilipczuk, M., Wojtaszczyk, J.: On multiway cut parameterized above lower bounds. In: Proceedings of the 6th International Symposium on Parameterized and Exact Computation, IPEC 2011, *Lecture Notes in Computer Science*, vol. 7112, pp. 1–12. Springer Verlag (2011)

32. Dantsin, E., Hirsch, E.A.: Satisfiability certificates verifiable in subexponential time. In: Proceedings of the 14th International Conference on Theory and Applications of Satisfiability Testing, SAT 2011, *Lecture Notes in Computer Science*, vol. 6695, pp. 19–32. Springer Verlag (2011)
33. Dehne, F., Fellows, M., Langston, M., Rosamond, F., Stevens, K.: An $O(2^{O(k)}n^3)$ FPT algorithm for the undirected feedback vertex set problem. Theory Comput. Syst. **41**(3), 479–492 (2007)
34. Dell, H., Marx, D.: Kernelization of packing problems. In: Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, pp. 68–81 (2012)
35. Dell, H., van Melkebeek, D.: Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. In: Proceedings of the 42nd Annual Symposium on Theory of Computing, STOC 2010, pp. 251–260 (2010)
36. Demaine, E.D., Hajiaghayi, M.: The bidimensionality theory and its algorithmic applications. Comput. J. **51**(3), 292–302 (2008)
37. Demaine, E.D., Hajiaghayi, M., Thilikos, D.M.: The bidimensional theory of bounded-genus graphs. SIAM J. Discrete Math. **20**(2), 357–371 (2006)
38. Demaine, E.D., Hajiaghayi, M.T.: Bidimensionality: new connections between FPT algorithms and PTASs. In: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005, pp. 590–601 (2005)
39. Dorn, F.: Planar subgraph isomorphism revisited. In: Proceedings 27th International Symposium on Theoretical Aspects of Computer Science, STACS 2010, *Dagstuhl Seminar Proceedings*, vol. 5, pp. 263–274. Leibniz-Zentrum für Informatik, Schloss Dagstuhl, Germany (2010)
40. Dorn, F., Fomin, F.V., Lokshtanov, D., Raman, V., Saurabh, S.: Beyond bidimensionality: Parameterized subexponential algorithms on directed graphs. In: Proceedings 27th International Symposium on Theoretical Aspects of Computer Science, STACS 2010, *Dagstuhl Seminar Proceedings*, vol. 5, pp. 251–262. Leibniz-Zentrum für Informatik, Schloss Dagstuhl, Germany (2010)
41. Downey, R.: A basic parameterized complexity primer. This volume.
42. Downey, R.G., Fellows, M.R.: Fixed-parameter tractability and completeness. In: Proceedings of the Twenty-first Manitoba Conference on Numerical Mathematics and Computing (Winnipeg, MB, 1991), vol. 87, pp. 161–178 (1992)
43. Downey, R.G., Fellows, M.R.: Parameterized computational feasibility. In: P. Clote, J. Remmel (eds.) Proceedings of the Second Cornell Workshop on Feasible Mathematics, Feasible Mathematics II, pp. 219–244. Birkhauser Boston (1995)
44. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Monographs in Computer Science. Springer, New York (1999)
45. Eppstein, D.: Subgraph isomorphism in planar graphs and related problems. J. Graph Algorithms Appl. **3**(3) (1999)
46. Feige, U.: Faster FAST (feedback arc set in tournaments). CoRR **abs/0911.5094** (2009)
47. Fellows, M.R., Fomin, F.V., Lokshtanov, D., Losievskaja, E., Rosamond, F.A., Saurabh, S.: Distortion is fixed parameter tractable. In: Proceedings of the 36th International Colloquium on Automata, Languages and Programming, ICALP 2009, Part I, *Lecture Notes in Computer Science*, vol. 5555, pp. 463–474. Springer Verlag (2009)
48. Fellows, M.R., Fomin, F.V., Lokshtanov, D., Rosamond, F.A., Saurabh, S., Szeider, S., Thomassen, C.: On the complexity of some colorful problems parameterized by treewidth. Inf. Comput. **209**(2), 143–153 (2011)

49. Flum, J., Grohe, M.: The parameterized complexity of counting problems. SIAM J. Comput. **33**(4), 892–922 (2004)
50. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Berlin (2006)
51. Fomin, F.V., Golovach, P.A., Lokshtanov, D., Saurabh, S.: Algorithmic lower bounds for problems parameterized with clique-width. In: Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, pp. 493–502 (2010)
52. Fomin, F.V., Kratsch, D.: Exact Exponential Algorithms, 1st edn. Springer (2010)
53. Fomin, F.V., Pilipzuk, M.: Jungles, bundles, and fixed parameter tractability. CoRR **abs/1112.1538** (2011)
54. Fomin, F.V., Villanger, Y.: Subexponential parameterized algorithm for minimum fill-in. In: Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, pp. 1737–1746 (2012)
55. Fortnow, L., Santhanam, R.: Infeasibility of instance compression and succinct PCPs for NP. In: Proceedings of the 40th Annual Symposium on Theory of Computing, STOC 2008, pp. 133–142 (2008)
56. Ganian, R., Hlinený, P., Kneis, J., Meister, D., Obdrzálek, J., Rossmanith, P., Sikdar, S.: Are there any good digraph width measures? In: Proceedings of the 5th International Symposium on Parameterized and Exact Computation, IPEC 2010, *Lecture Notes in Computer Science*, vol. 6478, pp. 135–146. Springer Verlag (2010)
57. Gramm, J., Niedermeier, R., Rossmanith, P.: Fixed-parameter algorithms for closest string and related problems. Algorithmica **37**(1), 25–42 (2003)
58. Guillemot, S.: FPT algorithms for path-transversal and cycle-transversal problems. Discrete Optimization **8**(1), 61–71 (2011)
59. Guo, J., Gramm, J., Hüffner, F., Niedermeier, R., Wernicke, S.: Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization. J. Comput. System Sci. **72**(8), 1386–1396 (2006)
60. Hertli, T.: 3-SAT faster and simpler — Unique-SAT bounds for PPSZ hold in general. In: Proceedings of the 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, pp. 277–284 (2011)
61. Hüffner, F., Niedermeier, R., Wernicke, S.: Techniques for practical fixed-parameter algorithms. Comput. J. **51**(1), 7–25 (2008)
62. Hunter, P., Kreutzer, S.: Digraph measures: Kelly decompositions, games, and orderings. Theor. Comput. Sci. **399**(3), 206–219 (2008)
63. Impagliazzo, R., Paturi, R., Zane, F.: Which problems have strongly exponential complexity? J. Comput. System Sci. **63**(4), 512–530 (2001)
64. Johnson, T., Robertson, N., Seymour, P.D., Thomas, R.: Directed tree-width. J. Comb. Theory, Ser. B **82**(1), 138–154 (2001)
65. Kaplan, H., Shamir, R., Tarjan, R.E.: Tractability of parameterized completion problems on chordal, strongly chordal, and proper interval graphs. SIAM J. Comput. **28**(5), 1906–1922 (1999)
66. Kawarabayashi, K.: Planarity allowing few error vertices in linear time. In: Proceedings of the 50th Annual Symposium on Foundations of Computer Science, FOCS 2009, pp. 639–648 (2009)
67. Khot, S., Raman, V.: Parameterized complexity of finding subgraphs with hereditary properties. Theor. Comput. Sci. **289**(2), 997–1008 (2002)
68. Khot, S., Regev, O.: Vertex cover might be hard to approximate to within $2-\varepsilon$. In: 18th Annual IEEE Conference on Computational Complexity (CCC'03), pp. 371–378 (2003)

69. Kratsch, S.: Co-nondeterminism in compositions: A kernelization lower bound for a Ramsey-type problem. In: Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, pp. 114–122 (2012)
70. Kratsch, S., Pilipczuk, M., Pilipczuk, M., Wahlström, M.: Fixed-parameter tractability of multicut in directed acyclic graphs. CoRR **abs/1202.5749** (2012)
71. Kreutzer, S., Ordyniak, S.: Digraph decompositions and monotonicity in digraph searching. Theor. Comput. Sci. **412**(35), 4688–4703 (2011)
72. Kreutzer, S., Tazari, S.: Directed nowhere dense classes of graphs. In: Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, pp. 1552–1562 (2012)
73. Langston, M.A.: Fixed-parameter tractability, a prehistory. This volume.
74. Lokshtanov, D., Marx, D., Saurabh, S.: Known algorithms on graphs of bounded treewidth are probably optimal. In: Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, pp. 777–789 (2011)
75. Lokshtanov, D., Marx, D., Saurabh, S.: Slightly superexponential parameterized problems. In: Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, pp. 760–776 (2011)
76. Lokshtanov, D., Misra, N., Saurabh, S.: Kernelization – preprocessing with a guarantee. This volume.
77. Ma, B., Sun, X.: More efficient algorithms for closest string and substring problems. SIAM J. Comput. **39**(4), 1432–1443 (2009)
78. Marx, D.: Parameterized graph separation problems. Theoret. Comput. Sci. **351**(3), 394–406 (2006)
79. Marx, D.: On the optimality of planar and geometric approximation schemes. In: 48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07), pp. 338–348 (2007)
80. Marx, D.: Closest substring problems with small distances. SIAM Journal on Computing **38**(4), 1382–1410 (2008)
81. Marx, D.: Can you beat treewidth? Theory of Computing **6**(1), 85–112 (2010)
82. Marx, D., Razgon, I.: Fixed-parameter tractability of multicut parameterized by the size of the cutset. In: Proceedings of the 43rd Annual Symposium on Theory of Computing, STOC 2011, pp. 469–478 (2011)
83. Marx, D., Schlotter, I.: Obtaining a planar graph by vertex deletion. Algorithmica **62**, 807–822 (2012)
84. Misra, N., Raman, V., Saurabh, S.: Lower bounds on kernelization. Discrete Optimization **8**(1), 110–128 (2011)
85. Monien, B.: How to find long paths efficiently. In: Analysis and design of algorithms for combinatorial problems (Udine, 1982), *North-Holland Math. Stud.*, vol. 109, pp. 239–254. North-Holland, Amsterdam (1985)
86. Nemhauser, G.L., Trotter Jr., L.E.: Vertex packings: structural properties and algorithms. Math. Programming **8**, 232–248 (1975)
87. Niedermeier, R.: Invitation to fixed-parameter algorithms, *Oxford Lecture Series in Mathematics and its Applications*, vol. 31. Oxford University Press, Oxford (2006)
88. Patrascu, M., Williams, R.: On the possibility of faster SAT algorithms. In: Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, pp. 1065–1075 (2010)
89. Paturi, R., Pudlák, P.: On the complexity of circuit satisfiability. In: Proceedings of the 42nd Annual Symposium on Theory of Computing, STOC 2010, pp. 241–250 (2010)
90. Raman, V., Saurabh, S.: Parameterized algorithms for feedback set problems and their duals in tournaments. Theor. Comput. Sci. **351**(3), 446–458 (2006)

91. Razgon, I., O'Sullivan, B.: Almost 2-SAT is fixed-parameter tractable. J. Comput. Syst. Sci. **75**(8), 435–450 (2009)
92. Reed, B., Smith, K., Vetta, A.: Finding odd cycle transversals. Operations Research Letters **32**(4), 299–301 (2004)
93. Robertson, N., Seymour, P.D.: Graph minors. XIII. The disjoint paths problem. J. Combin. Theory Ser. B **63**(1), 65–110 (1995)
94. Robertson, N., Seymour, P.D.: Graph minors. XX. Wagner's conjecture. J. Combin. Theory Ser. B **92**(2), 325–357 (2004)
95. Scheffler, P.: A practical linear time algorithm for disjoint paths in graphs with bounded tree-width. Tech. Rep. 396/1994, Technical University of Berlin (1994)
96. Thilikos, D.M.: Graph Minors and parameterized algorithm design. This volume.
97. Williams, R.: Finding paths of length k in $O^*(2^k)$ time. Inf. Process. Lett. **109**(6), 315–318 (2009)
98. Yap, C.K.: Some consequences of non-uniform conditions on uniform classes. Theor. Comput. Sci. **26**, 287–300 (1983)