Approximation algorithms and parameterized complexity are usually considered to be two separate ways of dealing with hard algorithmic problems. In this paper, our aim is to investigate how these two fields can be combined to achieve better algorithms than what any of the two theories could offer. We discuss the different ways parameterized complexity can be extended to approximation algorithms, survey results of this type and propose directions for future research.

Keywords: approximation algorithms; parameterized complexity

Received 12 March 2006; revised 11 December 2006

1. INTRODUCTION

Many of the computational problems that arise in practice are optimization problems: the task is to find a solution where the cost, quality, size, profit or some other measure is as large or small as possible. The NP-hardness of an optimization problem implies that, unless \( P = NP \), there is no polynomial-time algorithm that finds the exact value of the optimum.

Of course, the unfortunate fact that we cannot find the optimum in polynomial time does not give us an excuse to ignore the problem. After all, in practice, some solution is required. If we want to design and analyze algorithms for such problems in a mathematically rigorous way, then there are several options ahead of us. The field of exact algorithms relaxes the requirement that the running time is polynomial, and here our aim is to find the algorithm with the fastest running time, which is usually exponential in the size of the input. In parameterized complexity, the running time is analyzed in finer detail: instead of expressing it as a function of the input size, one or more parameters of the instance are defined, and we investigate the effect of these parameters on the running time. The goal is to design algorithms that work efficiently if the parameters of the input instance are small (even if the size of the input is large). When designing approximation algorithms, we relax the requirement that the algorithm produces an optimum solution, and our aim is to devise a polynomial-time algorithm such that the solution it produces is not necessarily optimal, but there is some worst-case bound on the solution quality.

The motivation for studying approximation algorithms is two fold. Firstly, if we have an approximation algorithm whose error guarantee is really good (say, the maximum error is 1%), then in practice it can be as good as an optimum solution. However, for many approximation algorithms in the literature, the error guarantee is much higher (50, 100, 1000, 10000%, or even worse). In this case, we cannot argue that this approximation algorithm is almost as good as an exact algorithm. Nevertheless, such algorithms are still important from the theoretical point of view, as they allow us to better understand and classify problems and their variants.

In the literature on approximation algorithms, the goal is almost always to find a polynomial-time approximation algorithm. However, the question of approximability makes sense also in the framework of parameterized complexity. By applying ideas from both theories, we might be able to tackle problems that are intractable to both theories. There can be problems that are both inapproximable and fixed-parameter intractable, but have parameterized approximation algorithms running in fixed-parameter tractable (FPT)-time.

The aim of this survey paper is to investigate the various ways the notion of approximability can appear in parameterized complexity. We review the results on this topic from the literature, and try to interpret these results in a common framework. To motivate further research, we discuss the different directions that can be pursued. The following four main issues are identified:

- **Approximation with instance parameters**: Consider an optimization problem, where an integer parameter \( k \) is associated with every input instance. Is it possible to find an approximation algorithm with running time \( f(k) \cdot |x|^{o(1)} \)? For example, in the **Partial Vertex Cover** problem, the task is to cover as many edges as
possible with $k$ vertices (where $k$ is part of the input). Is it possible to find a 2-approximation in time $f(k) \cdot |x|^{|k|/2}$?

Another example: GRAPH COLORING is fixed-parameter intractable if the parameter is the genus of the graph. However, there is a 2-approximation algorithm with running time $f(g) \cdot |x|$, if $g$ is the genus of the graph [1].

- **Approximation parameterized by cost**: The most direct application of parameterized complexity to optimization problems is to parameterize by the optimum value. That is, we try to find an exact algorithm that solves the problem in $f(OPT) \cdot |x|^{|OPT|}$ time, where OPT is the value of the optimum solution. The motivation is that such an algorithm can work efficiently if the optimum value is very small compared to the size of the input, which might be a reasonable assumption in certain applications. We can investigate the analogous question for approximation algorithms as well. For example, we might be interested in the question whether there is a 2-approximation algorithm for MAXIMUM CLIQUE with running time $f(OPT) \cdot |x|^{|OPT|}$.

- **Performance functions instead of performance ratios**: The usual aim of approximation theory is to find an algorithm that has a parameter $e$ such that the ratio of the optimum value and the running time is called an $FPT$-time algorithm. Analogously to NP-completeness in classical complexity, the theory of $W[1]$-hardness can be used to give strong evidence that certain problems are unlikely to be FPT. $W[1]$-hardness is usually proved by presenting a parameterized reduction. For the technical details, the reader is referred to [2, 3].

These four issues are not completely separate from each other. There are many ways in which they overlap and interact. However, we would like to emphasize here that parameterized approximation is a complex field, and there are many subareas to investigate.

Section 2 reviews the basic notions of parameterized complexity and approximation algorithms. The four issues are discussed in Sections 3–6. Conclusions are given in Section 7. Most of the results presented in this paper are taken from the literature. Some of the cited results are mentioned only informally, while others are presented as theorems. To maintain the flow of the text, only those results are highlighted as theorems that are directly related to both parameterization and approximability.

2. **PRELIMINARIES**

2.1 **Parameterized complexity**

We follow [2] for the standard definitions of parameterized complexity. Let $\Sigma$ be a finite alphabet. A decision problem is represented by a set $Q \subseteq \Sigma^*$ of strings over $\Sigma$. A parameterization of a problem is a polynomial-time computable function $\kappa : \Sigma^* \rightarrow N$. A parameterized decision problem is a pair $(Q, \kappa)$, where $Q \subseteq \Sigma^*$ is an arbitrary decision problem and $\kappa$ is a parameterization.\footnote{Some authors (e.g., [3]) prefer to define parameterized problems such that the parameter value is not a function of the instance, but it is a number explicitly given in the instance. Each approach has its strengths and weaknesses. We prefer to define the parameter as a function, to emphasize that a problem can have many possible parameterizations.}

Intuitively, we can imagine a parameterized problem as being a decision problem where each input instance $x \in \Sigma^*$ has a positive integer $\kappa(x)$ associated with it. A parameterized problem $(Q, \kappa)$ is $\text{FPT}$ if there is an algorithm that decides whether $x \in Q$ in time $f(\kappa(x)) \cdot |x|^c$ for some constant $c$ and computable function $f$. An algorithm with such running time is called an $\text{FPT}$-time algorithm or simply $\text{FPT}$-algorithm. The theory can be extended to problems having multiple parameterizations; in this case, we say that a problem $(Q, \kappa_1, \ldots, \kappa_p)$ is $\text{FPT}$ with combined parameters $\kappa_1, \ldots, \kappa_p$ if it can be solved in time $f(\kappa_1(x), \ldots, \kappa_p(x)) \cdot |x|^e$.

Many NP-hard problems were investigated in the parameterized complexity literature, with the goal of identifying FPT problems. It turns out that several of these problems, e.g., MINIMUM VERTEX COVER, LONGEST PATH, DISJOINT TRIANGLES, etc., are indeed FPT. There is a powerful toolbox of techniques for designing FPT-algorithms: kernelization, bounded search trees, color coding, well-quasi-ordering, just to name some of the more important ones. On the other hand, certain problems resisted every attempt at obtaining FPT-algorithms. Analogously to NP-completeness in classical complexity, the theory of $W[1]$-hardness can be used to give strong evidence that certain problems are unlikely to be FPT. $W[1]$-hardness is usually proved by presenting a parameterized reduction.
defined for each feasible solution. The task is to find a feasible solution where the measure is as good as possible. Following [4], an NP optimization problem is formally defined as a 4-tuple $(I, \text{sol}, \text{cost}, \text{goal})$, where

- $I$ is the set of instances;
- for an instance $x \in I$, $\text{sol}(x)$ is the set of feasible solutions of $x$. The length of each $y \in \text{sol}(x)$ is polynomially bounded in $|x|$, and it can be decided in polynomial time whether $y \in \text{sol}(x)$ holds for given $x$ and $y$;
- given an instance $x$ and a feasible solution $y$, $\text{cost}(x,y)$ is a polynomial-time computable positive integer;
- goal is either min or max.

The goal is to find a feasible solution $y$ that achieves the best objective value, i.e.

$$\text{cost}(x, y) = \text{goal} \left[ \text{cost}(x, y') : y' \in \text{sol}(x) \right].$$

The cost of the optimum solution for instance $x$ is denoted by $\text{opt}(x)$. If $y$ is a solution for instance $x$, then the performance ratio of $y$ is defined as

$$R(x, y) = \begin{cases} \frac{\text{cost}(x, y)}{\text{opt}(x)}, & \text{if goal = min,} \\ \frac{\text{opt}(x)}{\text{cost}(x, y)}, & \text{if goal = max}. \end{cases}$$

Thus, $R(x, y)$ is always at least 1; the closer it is to 1, the closer the solution is to the optimum. For a real number $c > 1$, we say that an algorithm is a $c$-approximation algorithm, if it always produces a solution with performance ratio at most $c$.

### 2.3 Approximation schemes

It is a very common situation that after finding the first constant-factor approximation for some problem, improved algorithms with better and better approximation ratios are published in subsequent papers. For certain problems, an endless series of improvements is possible, as there is no 'best' approximation ratio. We say that a problem $X$ admits a PTAS if for every $\epsilon > 0$, there is a polynomial-time $(1 + \epsilon)$-approximation algorithm for $X$. More precisely, we want this to hold in a uniform way: there is one algorithm that can produce an arbitrary good approximation. That is, there is an algorithm $\mathcal{A}$ such that given an instance $x$ of $X$ and an $\epsilon > 0$, $\mathcal{A}$ produces a $(1 + \epsilon)$-approximate solution in time $|x|^{O(1/\epsilon)}$ for some function $f$. Clearly, such an algorithm runs in polynomial time for every fixed value of $\epsilon$. Approximation schemes are very abundant for geometric problems, but the literature contains many examples for other types of problems as well, e.g. for scheduling, packing or pattern matching problems.

If $\epsilon$ is small then the exponent of the polynomial $|x|^{O(1/\epsilon)}$ can be very large. Two restricted classes of approximation schemes were defined that avoid this problem. An efficient PTAS EPTAS is a PTAS with running time of the form $f(1/\epsilon) \cdot |x|^{O(1)}$, while a fully PTAS FPTAS runs in time $(1/\epsilon)^{O(1)} \cdot |x|^{O(1)}$.

It is possible to prove negative results on the existence of approximation schemes using APX-hardness. APX is the class of optimization problems that have constant-factor approximation algorithms. A PTAS for an APX-hard problem would imply that there is a PTAS for every problem in APX. The celebrated PCP theorem (cf. [4]) states that this would also imply $P = NP$. Consequently, for every APX-hard optimization problem there exists a constant $c_0 > 1$ such that there is no polynomial-time $c_0$-approximation algorithm for the problem, unless $P = NP$. Often, there is a large gap between this lower bound $c_0$ and the best known approximation ratio. However, for the problem Max E3SAT, a tight bound of $(\frac{7}{6})$ is proved by Håstad [5].

### 2.4 Decision problems

In many cases, it is easier to work with decision problems than with optimization problems: most of complexity theory is developed for decision problems. There is a standard way in which an optimization problem can be turned into a more or less equivalent decision problem. If $X$ is an optimization problem, then we define decision problem $X_{\leq}$ as

$$X_{\leq}$$

Input: An instance $x$ of $X$, an integer $k$.

Decide: $\text{OPT}(x) \leq k$.

The problems $X_{\leq}$ and $X_{=}$ can be defined analogously. If $X$ can be solved optimally in polynomial time, then clearly all these decision problems can be solved in polynomial-time as well. The other direction is not that much clear. If we can solve $X_{\leq}$ in polynomial time, then $\text{OPT}(x)$ can be determined in polynomial time using binary search, but this does not give us a solution with optimum cost. However, for many problems, if we can determine the optimum, then we can actually find an optimum solution by repeatedly determining the optimum cost for slightly modified versions of the instance, a strategy generally known as polynomial-time self-reducibility.

In parameterized complexity, the problems $X_{\leq}, X_{=}, X_{=}$ are studied with the parameter $k$. If $X$ is a minimization (respectively, maximization) problem, then the standard parameterization of $X$ is the problem $X_{\leq}$ (respectively, $X_{=}$) parameterized by $k$. If the standard parameterization is FPT, then this means that we have an efficient algorithm for determining the optimum for those instances where the optimum is small.
3. APPROXIMATION WITH INSTANCE PARAMETERS

The first way of studying parameterized approximation algorithms that we discuss here is to define a parameterization of the optimization problem, and instead of looking for a polynomial-time algorithm, our goal is to find an FPT-time algorithm that produces an approximate solution. That is, we define a parameterization \( \kappa \) that assigns a positive integer to each instance \( x \in I \), and the approximation algorithm should run in time \( f(\kappa(x)) \cdot |x|^{O(1)} \). In this case, we say that the algorithm is an FPT-\( \kappa \)-approximation algorithm with parameterization \( \kappa \). If the performance ratio of the algorithm is \( c \), then we say that it is an \( FPT \) \( c \)-approximation algorithm. Hopefully, such an algorithm can achieve a better approximation ratio than traditional polynomial-time approximation algorithms, as it has more time at its disposal.

The following two results are examples of this type of parameterized approximation algorithms (definitions will be given later).

- **Metric TSP with Deadlines (\( \Delta DTSP \))** can be 2.5-approximated in time \( O(|x|^3 + k|k|) \), where \( k \) is the number of deadline vertices [6].
- **Vertex Coloring** can be 2-approximated in \( f(g) \cdot |x| \) time for graphs with genus \( g \) [1].

Broadly speaking, there are two large categories of parameterizations: the parameter is either some obvious measure of the problem instance (e.g. the number of deadline vertices in \( \Delta DTSP \)) or it is some structural property of the input structure that describes (in some well-defined sense) how complicated the input is (e.g. tree width, genus, etc.). In both cases, it can be interesting to find approximation algorithms that are efficient for small parameter values. Sections 3.1 and 3.2 discuss results of these two types, respectively.

Parameterized approximation schemes can be defined in a similar manner. An FPT-\( \kappa \)-approximation scheme (FPT-AS) with parameterization \( \kappa \) is an algorithm whose input is an instance \( x \in I \) and an \( \epsilon > 0 \), and it produces a (1 + \( \epsilon \))-approximate solution in time \( f(\epsilon, \kappa(x)) \cdot |x|^{O(1)} \) for some computable function \( f \). For example, Har-Peled and Mazumdar [7] present an \( f(\epsilon, k, d) \cdot |x| \) time approximation scheme for the Minimum \( k \)-Median problem in \( d \) dimensions, which means that the problem has an FPT-AS with combined parameters \( k \) and \( d \). This can be considered as a generalization of EPTAS: the constant factor depends not only on \( \epsilon \), but also on the parameters of the instance. Sections 3.1 and 3.2 review this type of approximation results as well.

3.1. Measure parameters

In this section, we overview parameterized approximation results where the parameter is some obvious measure of the input instance.

The traveling salesperson problem (TSP) is one of the most studied optimization problems. The input consists of \( n \) cities and a matrix describing the distances between the cities (the distances can be arbitrary nonnegative integers). The task is to start from one of the cities, visit every city in some order and then return to the starting city. We have to find an ordering of the cities such that the length of this tour is as small as possible. The problem is NP-hard, in fact, there is no \( c \)-approximation algorithm for any \( c \geq 1 \) (unless \( P = NP \)). The Metric TSP is a restricted version of TSP where we assume that the distance matrix satisfies the triangle inequality \( d(x, y) \leq d(x, z) + d(z, y) \) for any three cities \( x, y, z \). This assumption obviously holds in applications where we know that the direct route is never longer than a route via multiple cities. Metric TSP can be 1.5-approximated using Christofides’ algorithm [8], but has no PTAS, unless \( P = NP \) [9].

**Theorem 3.1** [6]. **Metric TSP with Deadlines** (\( \Delta DTSP \)) is investigated in [6]. Here we have the further restriction that some subset \( D \) of the vertices have deadlines assigned to them, and we have to visit each such vertex before its deadline. That is, each \( v \in D \) has a deadline \( d(v) \), and the length of the tour from the starting city to \( v \) must not exceed \( d(v) \). The task is to find the shortest tour that satisfies all the deadlines. The problem has no polynomial-time constant factor approximation algorithm, unless \( P = NP \). Furthermore, it is not FPT if the parameter is the number \( |D| \) of deadline vertices; in fact, it is NP-hard even if there is only one deadline vertex. Thus, neither approximation nor parameterized complexity alone can tackle this problem. However, [6] presents a \( O(|x|^3 + k|k|) \) time algorithm that produces a 2.5-approximate solution, if \( k \) is the number of deadline vertices. Therefore, the combination of approximation and parameterized complexity results in an efficient approximation algorithm for small values of \( k \).

**Theorem 3.1** [6]. **Metric TSP with Deadlines parameterized by the number of deadline vertices has an FPT 2.5-approximation algorithm.**

The **Vertex Cover** problem (cover all the edges of a graph with as few vertices as possible) is well studied both in the parameterized complexity and approximation algorithms literature. **Vertex Cover** is FPT (see, e.g. [10, 11]) and has a simple 2-approximation algorithm. However, the problem does not have a PTAS (unless \( P = NP \)); in fact, some recent conjectures imply that the factor 2 is best possible [12].

**Partial Vertex Cover** is in some sense the dual of **Vertex Cover**: given a graph \( G \) with an integer \( k \), the task is to cover as many edges as possible with \( k \) vertices. It can be shown that this problem is \( W[1] \)-hard with parameter \( k \) [13]. The straightforward greedy algorithm (repeatedly select a vertex that covers as many uncovered edges as possible) gives a 1.582-approximation [14], but the problem does not admit a PTAS [15]. Here we show that the problem admits an FPT-AS.
Theorem 3.2. Partial Vertex Cover admits an FPT-AS with parameter \( k \), the number of vertices in the solution.

Proof. We present an \( f(\varepsilon,k) \cdot x_{\text{OPT}}^{O(1)} \) time algorithm that produces a \( (1 + \varepsilon) \)-approximate solution for the problem. Let \( D := 2\left(\frac{k}{2}\right) / \varepsilon \) and let \( v_1, \ldots, v_n \) be the vertices of the graph ordered by non-increasing degree, i.e. \( d(v_i) \geq d(v_j) \) for \( i < j \).

We consider two cases.

Case 1: \( d(v_i) \geq D \). In this case the algorithm outputs the set \( S = \{v_1, \ldots, v_k\} \). These \( k \) vertices cover at least \( \sum_{i=1}^{k} d(v_i) - \left(\frac{k}{2}\right) \) edges: there are at most \( \left(\frac{k}{2}\right) \) edges between the vertices of \( S \), thus at most \( \left(\frac{k}{2}\right) \) edges are counted twice when the degrees are summed. The value of the optimum solution cannot be larger than \( \sum_{i=1}^{k} d(v_i) \), hence the value of the constructed solution \( S \) is at least

\[
\frac{\sum_{i=1}^{k} d(v_i) - \frac{k}{2}}{D} \geq 1 - \frac{\frac{k}{2}}{D} = 1 - \frac{\varepsilon}{2} \cdot \frac{1}{1 + \varepsilon}
\]

times the optimum, that is, the performance ratio is at most \( 1 + \varepsilon \).

Case 2: \( d(v_i) \leq D \). In this case, the optimum value is at most \( kD \), and we are able to determine it exactly in FPT-time. For each \( 1 \leq \ell \leq kD \), the algorithm checks whether it is possible to cover at least \( \ell \) edges with \( k \) vertices. We use the method of color coding: random colors between 1 and \( \ell \) are assigned to the edges, and we try to find a solution where the \( \ell \) edges covered by the vertices have distinct colors. As shown below, the existence of such a solution can be checked in FPT-time for a given coloring. If it is not possible to cover \( \ell \) edges, then obviously the algorithm does not find a solution for any random coloring. On the other hand, if there is a solution, then the random coloring assigns distinct colors to the \( \ell \) edges with probability at least \( \ell^{-\varepsilon} \). Therefore, we have to repeat the algorithm on average at most \( \ell^{\varepsilon} \) times to find a solution. Note that \( \ell^{\varepsilon} \) can be bounded by a function of \( \varepsilon \) and \( k \) only. The algorithm can be derandomized using standard techniques, (see [2, Section 13.3; 3, Section 8.3; 16]).

For a particular coloring, we proceed as follows. We consider every possible partition \( \mathcal{P} = \{P_1, \ldots, P_k\} \) of the color set \( \{1, \ldots, \ell\} \) into \( k \) classes; there are at most \( k^k \) such partitions. For a given partition \( \mathcal{P} \), we try to find vertices \( u_1, \ldots, u_k \) such that for each color \( c \in P_i \), vertex \( u_i \) covers at least one edge with color \( c \). This can easily be done in polynomial time.

One of the most fruitful areas for developing approximation algorithms is the field of geometric problems. The geometric nature of the problem allows us to use all sorts of approximation techniques. Very often, the problem even admits a PTAS. The computational geometry community has been doing parameterized approximation analysis for several years: the quest for obtaining approximation algorithms where certain constants do not appear in the exponent of the input size is evident in the literature.

The \textsc{Minimum} \( k \)-\textsc{center} problem arises as a fundamental problem in many applications such as facility location, clustering and information retrieval. Given a set of points in the plane and an integer \( k \), the task is to select \( k \) center points such that each input point is close to some selected point. More precisely, we have to minimize the maximum distance of a point to the closest selected point.

**Minimum \( k \)-center**

Input: A set \( S \subseteq \mathbb{R} \times \mathbb{R} \) of points, an integer \( k \).
Find: A subset \( C \subseteq S \) of size \( k \).
Goal: Minimize \( \max_{c \in C} \min_{s \in S} d(s,c) \).

There are several ways of interpreting distance in the plane. Here we discuss the two most common cases, where the distance means \( L_2 \) metric \( d((x_1, y_1), (x_2, y_2)) = \sqrt{|x_1 - x_2|^2 + |y_1 - y_2|^2} \) or \( L_\infty \) metric \( d((x_1, y_1), (x_2, y_2)) = \max(|x_1 - x_2|, |y_1 - y_2|) \). In both cases, the problem can be \( 2 \)-approximated in polynomial time, but there is some constant \( \alpha > 2 \) such that it is \( \alpha \)-approximation [17]. If the number of centers is a fixed constant \( k \), then the problem can be solved exactly in time \( |x|^{O(k)} \) by brute force. This cannot be improved to \( f(k) \cdot |x|^{O(1)} \) time: parameterized by the number \( k \) of centers, the problem is \( W[1] \)-hard in the \( L_\infty \) metric [18]. However, there are efficient approximations for every fixed value of \( k \); [19] shows that for every fixed \( k \) there is a linear-time approximation scheme, i.e. a \( (1 + \varepsilon) \)-approximate solution can be found in \( f(\varepsilon, k) \cdot n \) time. Using the terminology of parameterized approximation, this can be stated as follows.

Theorem 3.3. [19]. \textsc{Minimum} \( k \)-\textsc{center} admits an FPT-AS with parameter \( k \).

\textsc{Minimum} \( k \)-\textsc{median} is the variant of the problem where instead of minimizing the maximum distance to the closest center point, the optimization goal is to minimize the sum of these distances. Unlike \textsc{Minimum} \( k \)-\textsc{center}, this problem admits an EPTAS in the \( L_2 \) metric (even if \( k \) is part of the input) [7].

Most geometric problems can easily be generalized to higher dimensions. If the dimension of the input can be arbitrarily large, then usually the problem becomes as hard as on any general metric, since we lose the nice geometric properties that make approximation easy (cf. [20]). However, in many cases the geometric approximation schemes can be generalized to any fixed dimension \( d \). The approximation schemes for \textsc{Minimum} \( k \)-\textsc{center} [19] and for \textsc{Minimum} \( k \)-\textsc{median} [7] work for an arbitrarily fixed dimension \( d \),
and the running time is \( f(e,k,d) \cdot |x|^{O(1)} \) and \( f(e,d) \cdot |x|^{O(1)} \), respectively.

**Theorem 3.4** [7, 19]. Minimum \( k \)-median admits an FPT-AS with parameter \( d \). Minimum \( k \)-center admits an FPT-AS with combined parameters \( d \) and \( k \).

Dimension is a very natural parameter for the study of geometric problems. In the examples above, FPT-algorithms are possible if the dimension is taken as parameter. Conceivably, there can be problems where it is possible to show that the exponent of the input size has to increase as the dimension increases. There is very little work done in the literature on proving hardness results for problems parameterized by the dimension of the space. The only such result that we are aware of is on the Subset Congruence problem. Given two sets of points \( A, B \subseteq \mathbb{R}^d \), the task is to decide if there is a distance-preserving transformation that makes \( A \) equal to a subset of \( B \). The problem is polynomial-time solvable for every fixed \( d \), but \( \mathrm{W}[1] \)-hard parameterized by the dimension \( d \) [21].

### 3.2. Structural parameters

In the previous section, we discussed parameters that measure the size of some part of the input or the solution. These parameters were ‘obvious.’ Parameterization can be more subtle than that: it can measure arbitrary properties of the input structures. In many cases, the parameter measures how complicated the input is, in some precise, well-defined sense. The motivation is to obtain algorithms that are efficient for less complex instances, where less complex means that the parameter value is small. Such an algorithm can be useful in applications where it is a reasonable assumption that the input is not completely unstructured, and the value of this particular parameter is usually small.

Below we list some parameters that describe complexity in graphs. The list is not exhaustive; depending on the problem or application, many such parameters can be defined.

- **Maximum degree.** The maximum degree of a graph is the maximum number of neighbors a vertex can have.
- **Diameter.** The diameter of a graph is the maximum distance between any two vertices.
- **Tree width.** The tree width is a parameter that describes how ‘tree-like’ the graph is. We omit the technical definition, see, e.g. [22] for more details. In many cases, problems on graphs with small tree width can be handled with dynamic programming techniques.
- **Genus.** The genus of a graph describes how close the graph is to being a planar graph. A graph has genus 0 if and only if it is planar (i.e. can be drawn on the sphere without crossing edges). A graph has genus at most \( k \) if it can be drawn on the sphere with \( k \) handles attached to the sphere (cf. [23, Appendix B]).
- **Distance from a class \( \mathcal{F} \).** Let \( \mathcal{F} \) be an arbitrary class of graphs (such as planar, bipartite, interval, etc.) The class \( \mathcal{F} + \) respectively, \( \mathcal{F} + ke \) contains those graphs that can be constructed from some \( G \in \mathcal{F} \) with the addition of \( k \) new vertices\(^2\) (respectively, edges). The classes \( \mathcal{F} - kv \) and \( \mathcal{F} - ke \) are similarly defined. For any problem and any class \( \mathcal{F} \), we can assume that the input is in, say, \( \mathcal{F} + kv \) for some \( k \), and define this \( k \) to be the parameter. This line of research was initiated by Cai [24], and later pursued in, e.g. [25, 26].

The first problem that we investigate here is Vertex Coloring: given a graph \( G \), find a coloring of the vertices with minimal number of colors such that adjacent vertices receive different colors. The minimum number of colors that is required to color the vertices of \( G \) is called the chromatic number of \( G \) and is denoted by \( \chi(G) \). Chromatic number is not approximable within \( \lfloor V \rfloor^{1-\epsilon} \) for any \( \epsilon > 0 \), unless \( \mathsf{ZPP} = \mathsf{NP} \) [27]. The problem remains hard for planar graphs: it is \( \mathsf{NP} \)-complete to decide whether a planar graph is 3-colorable [28]. However, by the celebrated Four Color Theorem [29], four colors are sufficient for every planar graph, and there is a polynomial-time algorithm that actually finds this coloring [30]. As it is easy to check whether a graph is 2-colorable, a \( (\frac{1}{2}) \)-approximate coloring can be found in polynomial time for planar graphs.

Is there a constant factor approximation algorithm for graphs that are ‘almost’ planar graphs? We consider two classes of graphs that are close to being planar: bounded genus graphs and planar+kv graphs. For bounded genus graphs, the following result is implicit in [1]:

**Theorem 3.5** [1]. Vertex Coloring has an \( \mathsf{FPT} \) \( (\frac{1}{2}) \)-approximation algorithm if the parameter is the genus of the graph.

For planar+kv graphs, an easy brute force algorithm gives a \( (\frac{1}{\mathcal{F}}) \)-approximation:

**Theorem 3.6.** Vertex Coloring has an \( \mathsf{FPT} \) \( (\frac{1}{\mathcal{F}}) \)-approximation algorithm for planar+kv graphs.

**Proof.** Let \( X = \{v_1, \ldots, v_k\} \) be \( k \) vertices of \( G \) such that \( GX \) is planar (here we gloss over the question how this set is found or whether it is given in the input together with the planar+kv graph, cf. [31]). We can determine \( \chi(G[X]) \) by trying all the \( k^4 \) possible colorings of \( G[X] \). As discussed above, a \( (\frac{1}{\mathcal{F}}) \)-approximate coloring of the planar graph \( GX \) can be found in polynomial time. The coloring on \( G[X] \) and the coloring on \( GX \) can be combined to obtain a coloring of the whole graph with \( c = \chi(G[X]) + (\frac{1}{\mathcal{F}}) \chi(GX) \) colors. It is easy to check that \( c \leq (\frac{1}{\mathcal{F}}) \chi(G) \); the worst case occurs when \( \chi(G) = \)}
\(\chi(G[X]) = \chi(GX) = 3\), but the algorithm produces a coloring with \(3 + 4 = 7\) colors.

Note that Theorems 3.5 and 3.6 are incomparable: bounded genus graphs might not be planar + \(k\) for any \(k\) (e.g. large toroidal grids) and planar + \(1\) for graphs can have unbounded genus (e.g. a grid with an additional vertex connected to every vertex).

The **Maximum Independent Set** problem is NP-hard for planar graphs [32], but has a linear-time approximation scheme [33, 34]. It follows from [35] and [36] that this can be generalized to bounded-genus graphs.

**Theorem 3.7** [35, 36]. Maximum independent set admits a linear-time FPT-AS if the parameter is the genus of the graph.

For planar + \(k\) for graphs, a \((1 + \epsilon)\)-approximation of maximum independent set can be found by trying all \(2^k\) possibilities on the \(k\)-extra vertices, and then by finding a \((1 + \epsilon)\)-approximation on the remaining graph.

**Theorem 3.8.** **Maximum Independent Set** admits a linear-time FPT-AS on planar + \(k\) graphs.

In **Vertex Coloring**, the goal is to minimize the number of different colors used. In other words, if we identify the set of colors with the set of positive integers, then we try to minimize the maximum of the colors assigned. In **Minimum Sum Coloring**, the optimization goal is to minimize the sum of the colors (positive integers) on the vertices. Besides its combinatorial interest, the problem is motivated by applications in scheduling and VLSI design [37, 38]. The **Minimum Sum Edge Coloring** problem is defined analogously. The **Minimum Sum Coloring** is linear-time solvable on bounded tree width graphs [39], but the **Minimum Sum Edge Coloring** is one of the few problems that are polynomial-time solvable on trees [40, 41], but NP-hard already on graphs with tree width 2 [42]. However, **Minimum Sum Edge Coloring** has a linear-time PTAS for bounded tree width graphs (in fact, this is true even for the much more general multicoloring version).

**Theorem 3.9** [43]. **Minimum Sum Edge Coloring** admits an FPT-AS if the parameter is the tree width.

Note that without parameterization we cannot obtain an approximation scheme for **Minimum Sum Edge Coloring**, since the problem is APX-hard [42].

## 4. PARAMETERIZATION BY COST

In Section 3, our aim was to develop approximation algorithms that work efficiently if some parameter of the input is small. Perhaps the most obvious parameter of an optimization problem instance is the optimum cost. What we would like to have is an algorithm that efficiently finds an approximation of the optimum, if this optimum is small. Notice that many of the standard, well-studied problems in the parameterized complexity literature are standard parameterizations of certain optimization problems (e.g. **Minimum Vertex Cover**, **Maximum Clique**, **Minimum Dominating Set**, **Longest Path**, etc.) By studying the fixed-parameter tractability of these problems, we are investigating the possibility of having efficient exact algorithms for these problems in the case when the optimum is small. When a W[1]-hardness result shows that such an exact algorithm is unlikely to exist, then it is natural to study whether it is possible to approximate the optimum, if it is small.

Recently, at least three papers tried to extend parameterized complexity into this direction [44–46]. In Section 3, the definition of FPT-approximability (when the parameter is some property of the optimization instance) was a straightforward generalization of polynomial-time approximability. The definition becomes technically more delicate if we want to parameterize by cost. The first complication is that we have to decide whether we want to parameterize by the optimum cost (which is somewhat counterintuitive, since presumably the optimum cost is hard to determine), or we assume that the input contains a parameter \(k\) (the cost that should be reached), as in the standard parameterization of the problem. Here we follow the definition proposed by Chen et al. [45].

**Definition 4.1.** Let \(X = (I, \text{sol}, \text{cost}, \text{goal})\) be an optimization problem. A standard FPT-approximation algorithm with performance ratio \(c\) for \(X\) is an algorithm that, given an input \((x, k)\in\Sigma^*\times\mathbb{N}\) satisfying

\[
\begin{align*}
\text{OPT}(x) & \leq k, \quad \text{if goal = min}, \\
\text{OPT}(x) & \geq k, \quad \text{if goal = max}
\end{align*}
\]

computes a \(y \in \text{sol}(x)\) in time \(f(k)\cdot |x|^{o(1)}\) such that

\[
\begin{align*}
\text{cost}(x, y) & \leq k\cdot c, \quad \text{if goal = min}, \\
\text{cost}(x, y) & \geq k/c, \quad \text{if goal = max}.
\end{align*}
\]

For inputs not satisfying condition (1), the output can be arbitrary.

The word ‘standard’ signifies that this is the approximation version of the standard parameterization, and distinguishes it from the case when the parameter is unrelated to the cost (as in Section 3).

Unfortunately, we do not have a good example for this form of approximability, an example where a problem is not FPT, but has a standard FPT-approximation algorithm with some performance ratio \(c > 1\). Nevertheless, there are some examples of FPT-approximation algorithms appearing in the literature. Tree width is FPT, in fact for every \(k\), a tree decomposition of width \(k\) can be computed in linear time if it exists [47]. This algorithm is quite complicated and not
practical. However, there are much simpler FPT-algorithms that produce tree decompositions having width at most a constant factor larger than the optimum [48–57]. In the case of the rank width of graphs and branch width of matroids represented over a fixed-finite field, we have the curious situation that it can be decided in FPT-time whether the width is at most $k$ [52, 53], but it is not known whether a decomposition of width at most $k$ (if it exists) can be constructed in FPT-time. However, standard FPT-approximation algorithms exist for these problems [53–55].

One might be tempted to define the notion of standard FPT-AS to capture the situation where a problem has a standard FPT-approximation algorithm for optimization problem $X$ has a standard FPT-approximation (Section 3) can be extended similarly. We would like to have an analog of EPTAS, since approximation schemes where the exponent of input size increases as $\varepsilon$ decreases is not very interesting from the parameterized complexity point of view. Thus, the natural definition for a standard FPT-approximation scheme would be that for every $\varepsilon > 0$, there is a standard FPT-approximation algorithm with performance ratio $1 + \varepsilon$, and the running time is $f(k, e \cdot |x|^{O(1)})$. However, this notion is not very interesting to study, since it implies fixed-parameter tractability; therefore, it does not extend the class of problems that we can call tractable. The proof is essentially the same as the proof of Proposition 6.1.

**Proposition 4.1.** Assume that there is an algorithm that has an input parameter $\varepsilon$, and for every $\varepsilon > 0$ it produces a $(1 + \varepsilon)$-approximate solution for optimization problem $X$ in time $f(k, e \cdot |x|^{O(1)})$. Then the standard parameterization of $X$ is in FPT.

### 5. Non-constant Performance Functions

When designing approximation algorithms, the usual aim is to ensure that the cost of the solution differs from the optimum by at most a constant factor. This means that the algorithm has to perform equally well regardless of whether the optimum is ‘small’ or ‘large.’ The analysis can be made more precise if the performance of the algorithm is bounded by a function of the optimum cost, rather than by an absolute constant. This is especially natural in the case of standard FPT-approximation (Section 4): if the goal is to have an algorithm that is efficient for small cost values, then it makes sense to require good performance only if the optimum is small.

Let $g : \mathbb{N} \to \mathbb{N}$ be a nondecreasing function. We say that an approximation algorithm for optimization problem $X$ has **performance ratio function** $g(k)$ (or it is a $g(k)$-approximation algorithm) if it produces a solution with performance ratio at most $g(OPT(x))$ for every instance $x$. For example, if the solution has cost $OPT(x) \log OPT(x)$, then it is an log-approximation algorithm. Constant-factor approximability is the special case where $g(k)$ is a constant function. The definition of parameterized FPT-approximation (Section 3) can be extended similarly. To define standard FPT-approximability with performance ratio function $g(k)$, (2) has to be replaced with

$$
\begin{cases}
\text{cost}(x, y) \leq k \cdot g(k), & \text{if goal} = \text{min}, \\
\text{cost}(x, y) \geq k / g(k), & \text{if goal} = \text{max}.
\end{cases}
$$

If our goal is only to obtain an approximation algorithm with *some* performance function (no matter how bad it is), then in the case of maximization problems, FPT time does not give us more power than polynomial time. This surprising result was observed by Grohe and Gruber [56].

**Theorem 5.1.** If a maximization problem $X$ has a standard FPT-approximation algorithm with performance ratio function $g(k)$, then there is a polynomial-time $g'(k)$-approximation algorithm for $X$, for some function $g'(k)$.

**Proof.** Assume that $X$ has a standard FPT-approximation algorithm with running time $f(k) \cdot |x|^{O(1)}$ and performance ratio function $g(k)$. There is a computable nondecreasing function $b(n)$ such that every instance of $X$ with size at most $n$ has optimum at most $b(n)$. Denote by $c(n)$ the largest $i$ such that $i \leq n$, $f(i) \leq n$ and $f(i)$ can be computed in time $O(n)$; clearly, $c(n)$ is unbounded, nondecreasing and computable in time $n^{O(1)}$.

Given an instance $x$, the polynomial-time approximation algorithm proceeds as follows. For every $k = 1, 2, \ldots, \max(c(|x|), 1)$, we run the assumed FPT-approximation algorithm on $x$, and select the best solution returned by the different runs (since we try $k = 1$ and $OPT(x) \geq 1$ holds, some solution is always found). From $k \leq \max(c(|x|), 1)$ it follows that the $f(k) \cdot |x|^{O(1)}$ time FPT-algorithm runs in polynomial time, hence the running time of the whole procedure is polynomial in $|x|$.

If $c(|x|) \geq OPT(x)$, then the FPT-approximation algorithm is invoked with $k = OPT(x)$, which means that a solution of cost at least $OPT(x) / g(OPT(x))$ is produced. Assume therefore that $c(|x|) < OPT(x)$. Let $d(m)$ be the largest $n$ such that $b(n) < m$. This means that $|x| > d(OPT(x))$ for every instance $x$, hence $c(|x|) \geq c(d(OPT(x)))$. Thus, when the FPT-approximation algorithm is invoked with $k = c(d(OPT(x))) < OPT(x)$, then it produces a solution with value at least $r(OPT(x)) := c(d(OPT(x))) / g(c(d(OPT(x))))$. It is easy to see that $r(OPT(x))$ is an unbounded computable function of $OPT(x)$, hence the polynomial-time algorithm is an approximation algorithm with performance ratio function $g'(k) = k / r(k)$. \qed

We remark that the analog of Theorem 5.1 does not seem to hold for minimization problems.

### 5.1. Examples (polynomial-time)

Expressing the performance ratio of an algorithm as a function of the optimum can be interesting even in the case of polynomial-time approximation algorithms, independently of
parameterized complexity. We cite here three results of this type. Determining tree width is NP-hard [57], and no constant factor polynomial-time approximation algorithm is known (it is an important open question whether such an approximation is possible). However, the following approximation result was obtained recently by Feige et al. [58].

**Theorem 5.2** [58]. *Given a graph with tree width k, a tree decomposition of width at most O(k/\log k) can be constructed in polynomial time.*

That is, tree width can be \(O(\sqrt{\log k})\)-approximated in polynomial time. (The previous best result was a \(O(\log k)\)-approximation [51, 59].)

Whether **Minimum Directed Feedback Vertex Set** is in FPT or not is one of the most important open questions of parameterized complexity. In this problem, the task is to find a minimum set of vertices whose deletion makes the given directed graph acyclic. The undirected version of the problem is known to be in FPT (cf. [3 Section 8.1]). On the other hand, the directed version is \(\text{FPT}^\text{-hard} [45, 63]\), but apparently none of the techniques go through for the directed case. It seems that a much deeper understanding of the structure of directed graphs is required before this problem can be settled. However, it is observed in [45] that the results of [61, 62] imply an approximation algorithm with bounded performance ratio function.

**Theorem 5.3** [45]. *Minimum Directed Feedback Vertex Set has a polynomial-time approximation algorithm with performance ratio function \(O(\log k \cdot \log \log k)\).*

In the **Maximum Disjoint Cycles** problem, the task is to find \(k\) pairwise vertex disjoint cycles in a given graph. This problem is FPT (cf. [3 Section 8.1]). On the other hand, the directed version of the problem is \(\text{W}[1]-\text{hard} [45, 63]\), but polynomial-time solvable for every fixed value of \(k\) [64]. On the basis of the theoretical results proved in [64], Grohe and Gru¨ber [56] presented an FPT-approximation algorithm. By Theorem 5.1, the algorithm can be made to work in polynomial time.

**Theorem 5.4** [56]. *There is a polynomial-time approximation algorithm for Maximum Disjoint Directed Cycles with performance ratio function \(g(k)\), for some computable function \(g(k)\).*

The function \(g(k)\) is not given explicitly in [56], but \(k/g(k)\) is very slowly growing (even in the FPT-time version of the algorithm, before applying Theorem 5.1.).

### 5.2. Examples (FPT-time)

Graph layout problems arise in many application domains such as scheduling, VLSI design and archeology [65]. A *linear layout* of a graph \(G(V, E)\) is a one-to-one mapping \(\sigma\) between \(V\) and \([1, \ldots, |V|]\). The bandwidth of a layout is \(\max_{uv \in E} |\sigma(u) - \sigma(v)|\), the maximum ‘length’ of an edge in the layout. The *cutwidth* of a layout is \(\max_{j \in [1,|V|]} |\{u \in E : (u, v) \in E\}|\), the maximum number of edges that cross the cut formed by two neighboring vertices in the layout. The bandwidth (respectively cutwidth) of a graph is the minimum bandwidth (respectively cutwidth) over all possible linear layouts of the graph. Bandwidth is \(\text{W}[1]-\text{hard} [66]\), while cutwidth is \(\text{FPT} [67]\).

Subdividing an edge with a new degree two vertex might decrease the bandwidth of the graph. **Topological bandwidth** is the smallest bandwidth that can be achieved by repeatedly subdividing edges. Topological bandwidth is known to be \(\text{W}[1]-\text{hard} [68]\). Fellows [69] observed that if \(\text{tbw}(G)\) (respectively \(\text{cw}(G)\)) is the topological bandwidth (respectively cutwidth) of graph \(G\), then

\[\text{tbw}(G) \leq \text{cw}(G) + 1\]  

and

\[\text{cw}(G) < \text{tbw}(G)^2\]

holds. This means that if a graph has topological bandwidth at most \(k\), then one can obtain a layout with cutwidth less than \(k^2\) using the algorithm of [67]. Furthermore, using (3) (whose proof is algorithmic), this layout can be turned into a layout for a subdivision of \(G\) with bandwidth at most \(k^2\).

**Theorem 5.5** [69]. *Topological Bandwidth has a standard FPT-approximation algorithm with performance ratio function \(k\).*

The graph parameter *clique width* was introduced by Courcelle et al. [70]. Similarly to tree width, it measures the complexity of a graph with respect to certain composition operators. A *k*-expression describes a way of constructing the graph using these operations; the clique width of a graph is the smallest \(k\) such that it has a *k*-expression. Fellows et al. [71] have shown that determining clique width is NP-hard, in fact, it cannot be approximated with an absolute error guarantee, unless \(P = \text{NP}\). Oum and Seymour [54] use the approximability of rank width to obtain an approximation algorithm for clique width. They show that if \(\text{cw}(G)\) is the clique width and \(\text{rw}(G)\) is the rank width of a graph \(G\), then

\[\text{rw}(G) \leq \text{cw}(G) \leq 2^{\text{rw}(G) + 1} - 1\]

holds. The proof is algorithmic: given a rank decomposition of width \(k\), a \((2^{k+1} - 1)\)-expression can be obtained in polynomial time. There is an FPT-approximation algorithm that computes a rank decomposition of width \(3\text{rw}(G) + 1\) [55]. Therefore, clique width can be approximated by first computing an approximate rank decomposition and then turning it into a *k*-expression.
The significance of Theorem 5.6 comes from the fact that certain optimization problems are linear-time solvable for every $k$, if a $k$-expression of the graph is given in the input. That is, these problems are FPT parameterized by clique width, but only if the corresponding $k$-expression is given in the input. Theorem 5.6 can be used to remove the requirement that the $k$-expression is given in the input: if we know that the graph has a $k$-expression, then an $f(k)$-expression can be obtained in FPT-time, where $f(k) = 2^{k+2} - 1$. Then, instead of using the algorithm for $k$-expressions, we can use the algorithm for $f(k)$-expressions: the running time might be much larger, but it is still linear with a constant depending only on $k$. Therefore, obtaining the approximation and then solving the problem using the $f(k)$-expression can be done in FPT-time.

5.3. Negative results

For certain problems, it is possible to show that there is no standard FPT-approximation algorithm for any performance ratio function. Given a graph $G(V, E)$, an independent dominating set is an independent set $S \subseteq V$ of vertices such that for every $v \in V$, either $v \in S$, or $v$ is a neighbor of a member of $S$. The Minimum Independent Dominating Set problem is the corresponding optimization problem, where the goal is to minimize the size of the set $S$. Downey et al. [46] prove that this problem is completely inapproximable.

**Theorem 5.7** [46]. If Minimum Independent Dominating Set has a standard FPT-approximation algorithm with performance ratio function $\gamma(k)$ for some computable function $\gamma(k)$, then $W[2] = \text{FPT}$.

Chen et al. [45] presents inapproximability results for satisfiability problems. In the Min-WSAT(CIRC) problem, a Boolean circuit is given and the task is to find a satisfying assignment of minimum weight (where the weight of an assignment is the number of true variables).

**Theorem 5.8** [45]. If Min-WSAT(CIRC) has a standard FPT-approximation algorithm with performance ratio function $\gamma(k)$ for some computable function $\gamma(k)$, then $W[P] = \text{FPT}$.

Similar results hold for other variants of the satisfiability problem and for the parameterized halting problem [45].

All the inapproximability results presented above are somewhat unsatifying in the sense that the problems considered are not monotone. Monotone means that (in case of a minimization problem) if we extend a feasible solution with additional vertices/true variables, then it remains feasible. Clearly, this does not necessarily hold in these examples. Therefore, it can happen that the optimum is $k$, and every feasible solution has cost $k$, which makes approximation equivalent to finding an optimum solution. The inapproximability proofs in these examples tell us more about the hardness of finding exact solutions than about the hardness of approximation. It would be much more interesting (and possibly, more difficult) to have analogous inapproximability results for the monotone problems Maximum Clique and Minimum Dominating Set, for example.

6. Quality of approximation as parameter

A PTAS can produce solutions with approximation ratio arbitrary close to 1. However, we have to pay a price for that: the running time can ridiculously be large if $\epsilon$ is small. Table 1 (reproduced from [72]) presents the running time of some approximation schemes for $\epsilon = 0.2$. As we can see, running times of $O(|x|^{1.000,000})$ or worse is not uncommon if we require that the maximum error is 20%. Obviously, such algorithms are not useful in practice. Nevertheless, these results are important from the theoretical point of view, as they show that there is no lower bound on the approximation ratio that can be achieved in polynomial time.

The notion of EPTAS tries to formalize a restricted class of approximation schemes, where the algorithm might have a chance of being useful. If the running time of the algorithm is of the form $c^{1/\epsilon} \cdot n$, and $c$ is not too large, then the algorithm can be efficient for, say, $\epsilon = 0.2$. In many cases, the first approximation scheme obtained for a problem was not an EPTAS, but later it was improved to an EPTAS. For example, Arora [73] presented an $|x|^{O(1/\epsilon)}$ time PTAS for Euclidean TSP, which is not an EPTAS. However, in the journal version of the paper [74], the running-time of the algorithm is improved to $|x|^{O(1/\epsilon^2)} \cdot |x|^2$, hence the problem admits an EPTAS. Arora et al. [75] presented an $|x|^{O(1/\epsilon)}$ time approximation scheme for the Euclidean $k$-median problem. Later this was improved to an EPTAS with running time $2^{O(1/\epsilon \log 1/\epsilon)} \cdot n$ [7, 75]. In certain cases, the effort to obtain an EPTAS was only partially successful. For example, Hunt et al. [76] show that Maximum

**Table 1.** The running time of some PTASs with 20% error.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Reference</th>
<th>Running time for 20% error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Euclidean TSP</td>
<td>Arora [73]</td>
<td>$O(</td>
</tr>
<tr>
<td>Multiple Knapsack</td>
<td>Chekuri and Khanna [111]</td>
<td>$O(</td>
</tr>
<tr>
<td>Maximum Subforest</td>
<td>Shamir and Tsur [112]</td>
<td>$O(</td>
</tr>
<tr>
<td>Maximum</td>
<td>Erlebach et al. [113]</td>
<td>$O(</td>
</tr>
<tr>
<td>Independent Set for disk graphs</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
INDEPENDENT SET for unit disk graphs admits an $|x|^{O(1/e)}$ time PTAS. They are unable to present an EPTAS for the problem in general, but for the special case of $\lambda$-precision unit disk graphs (where the centers of the disk are not closer than $\lambda$ from each other) they give a $2^{O(1/e)*x^2} * n$ time EPTAS.

If a problem resists every attempt to obtain an EPTAS, then we should consider looking for some negative evidence showing that no EPTAS is possible, i.e. that it is not possible to get $1/e$ out of the exponent of the input size (modulo some complexity-theoretic assumption). It is not very surprising that parameterized complexity can provide such evidence, since getting out certain parameters from the exponent is the central issue of the theory. The basic tool that can be used to prove negative results for the existence of EPTASs is the following connection, observed independently by Bazgan [77] and Cesati and Trevisan [78].

**Proposition 6.1.** If an optimization problem $X$ admits an EPTAS, then the standard parameterization of $X$ is FPT.

**Proof.** Assume that we have an algorithm that produces a $(1 + \epsilon)$-approximate solution in time $f(1/\epsilon) \cdot |x|^{O(1)}$. Given an instance $(x, k)$ of the standard parameterization of $X$, we set $\epsilon := 1/(2k)$, and use the EPTAS to find a $(1 + \epsilon)$-approximate solution in time $f(1/\epsilon) \cdot |x|^{O(1)} = f(2k) \cdot |x|^{O(1)}$. Assume first that $X$ is a minimization problem. If the optimum is at most $k$, then the cost of the approximate solution is at most $(1 + \epsilon)k \leq k + 1/2 < k + 1$. As the cost is integer, the cost of the approximate solution is at most $k$. If the optimum is greater than $k$, then the cost of the approximate solution is also greater than $k$. Therefore, by checking whether the cost of the approximate solution is at most $k$, we can decide whether the optimum is at most $k$. The proof is similar in the case of maximization problems: if the optimum is at least $k$, then the cost of an $(1 + \epsilon)$-approximate solution is at least $k/(1 + \epsilon) = k - 1/(2 + 1/k) > k - 1$. □

We can use the contrapositive of Proposition 6.1 to show that it is unlikely that a particular problem admits an EPTAS.

**Corollary 6.1.** If the standard parameterization of an optimization problem is W[1]-hard, then the optimization problem does not have an EPTAS (unless FPT = W[1]).

It has to be remarked that the converse of Proposition 6.1 is not true. For example, MINIMUM VERTEX COVER is FPT, but it is APX-hard, hence it does not even have a PTAS. Therefore, Proposition has 6.1 limited applicability.

### 6.1. Examples

Table 2 lists problems where the existence of an EPTAS was ruled out using Proposition 6.1. The first four problems are geometric problems. For a set $V$ of geometric objects, the intersection graph of $V$ is a graph with vertex set $V$ where two vertices are connected if and only if the two objects have nonempty intersection. Intersection graphs of disks, rectangles, line segments and other objects arise in applications such as facility location [79], frequency assignment [80] and map labeling [81]. The first four lines in Table 2 show that if we restrict MAXIMUM INDEPENDENT SET and MINIMUM DOMINATING SET to the intersection graphs of unit-radius disks or unit-size squares, then the problem admits a PTAS [76, 82, 83], but does not have an EPTAS [18, 84]. COVERING POINTS WITH SQUARES is also a geometric problem, but it is not defined in terms of intersection graphs. In this problem, $n$ points are given in the plane, and the task is to cover all of them with at most $k$ (axis-parallel) unit squares. The squares can be at arbitrary locations in the plane.

**Choosing Substring Selection** and **Closest Substring** are two pattern matching problems that are motivated by applications in computational biology. In the **Closest Substring** problem, the task is to find a string of length $L$ that approximately appears in each of the $k$ input strings, where ‘approximately appears’ means that it appears with at most $d$ mismatches. The goal is to minimize this number $d$. In the **Distinguishing Substring Selection** problem the input strings are divided into ‘good’ strings and ‘bad’ strings, and we have to find a string of length $L$ that approximately appears in each of the good strings, but does not appear (even approximately) in any of the bad strings. We do not give a more detailed description of these problems, as many variants, parameterizations and optimization goals are defined in the literature, see [85–89].

Khanna et al. [90] defined classes of optimization problems that have PTASs. The key to approximability in these problems is the underlying planar structure, which allows us to
use Baker’s layering technique [34] and algorithms for bounded tree width. The problems are formulated using Boolean logical expressions. Recall that a formula in disjunctive normal form (DNF) is a disjunction of terms, e.g. \((x_1 \lor \overline{x}_2) \lor (x_1 \land \overline{x}_2 \land \overline{x}_3).\) A DNF is positive (respectively negative), if every literal is positive (respectively negated). The weight of an assignment is the number of variables that are set to true.

**MPSAT**

**Input:** A collection \(\mathcal{C} = \{\phi_1, \ldots, \phi_n\}\) of DNFs.

**Find:** An assignment \(\psi.\)

**Goal:** Maximize the number of DNFs satisfied by \(\psi.\)

**TMIN**

**Input:** A collection \(\mathcal{C} = \{\phi_1, \ldots, \phi_n\}\) of positive DNFs.

**Find:** An assignment \(\psi\) that satisfies every DNF in \(\mathcal{C}\)

**Goal:** Minimize the weight of \(\psi.\)

**TMAX**

**Input:** A collection \(\mathcal{C} = \{\phi_1, \ldots, \phi_n\}\) of negative DNFs.

**Find:** An assignment \(\psi\) that satisfies every DNF in \(\mathcal{C}\)

**Goal:** Maximize the weight of \(\psi.\)

These problems generalize many of the standard optimization problems: for example, it is easy to see that \(\text{Max Cut}\) can be reduced to \(\text{MPSAT};\) \(\text{Maximum Independent Set}\) can be reduced to \(\text{TMAX};\) and \(\text{Minimum Vertex Cover}\) can be reduced to \(\text{TMIN}.\) (In all three reductions, we associate a variable with each vertex and a DNF with each edge.) Cesati and Trevisan [78] study a problem of similar flavor, where the variables are not necessarily Boolean, they can have arbitrary domains. Let \(x_1, \ldots, x_n\) be variables and let \(D_i\) be the domain of variable \(x_i.\) A constraint over \(x_1, \ldots, x_n\) is an arbitrary relation over \(D_1, \ldots, D_n,\) i.e. each possible combination of values assigned to the variables either satisfies the constraint or not. A constraint is given in the input by listing the combination of values that satisfy the constraint.

**Multivalued Max ACSP**

**Input:** A collection \(\mathcal{C} = \{C_1, \ldots, C_m\}\) of constraints; each constraint is on \(k\) variables.

**Find:** An assignment \(\psi.\)

**Goal:** Maximize the number of constraints satisfied by \(\psi.\)

Given an instance of the above problems, the incidence graph is a bipartite graph defined by associating a vertex to each variable and each clause (constraint), and by connecting each variable to every clause (constraint) where it appears. Khanna and Motwani [91] show that \(\text{MPSAT}, \text{TMIN}, \text{TMAX}, \text{Multivalued Max ACSP}\) all admit a PTAS if the incidence graph is planar. The running time is \(\|x\|^{O(1/\epsilon)},\) hence it is not an EPTAS. Cesati and Trevisan [78] have shown that the standard parameterization of \(\text{Planar Multivalued Max 3CSP}\) is \(W[1]-hard,\) hence by Proposition, 6.1 the PTAS cannot be improved to an EPTAS, unless \(W[1] = \text{FPT}.\) Similar results were obtained for \(\text{MPSAT}, \text{TMIN}, \text{TMAX}\) by Cai et al. [92].

**Theorem 6.1** [78, 92]. The planar problems \(\text{MPSAT}, \text{TMIN}, \text{TMAX}, \text{Planar Multivalued Max 3CSP} do not admit an EPTAS, unless \(W[1] = \text{FPT}.\)**

### 6.2. Parameterization by \(1/\epsilon\)

All the negative results in Table 2 were obtained using Proposition 6.1. While this method is simple and apparently works, there is something unnatural about it. Our aim is to prove that \(1/\epsilon\) cannot be taken out of the exponent of the input size. In order to show this, we consider the standard parameterization, and prove that \(k\) cannot be taken out of the exponent. But it could not be done somehow more directly, by defining \(1/\epsilon\) to be the parameter? Intuitively, it is clear that what we are trying to determine is whether the problem is \(\text{FPT},\) with \(1/\epsilon\) being the parameter. However, it is not entirely clear what ‘defining \(1/\epsilon\) to be the parameter’ means. The problem is that the parameterized complexity theory is not developed for optimization problems, and optimization problems are usually studied via the standard parameterization. In the decision version, however, it is not immediately clear what the meaning of \(\epsilon\) is, as we do not have to produce a solution.

To make the idea of parameterization by \(1/\epsilon\) technically rigorous, we introduce a ‘gap’ version of the standard parameterization: the input comes with a promise stating that there is a gap between the yes-instances and the no-instances. Parameterization by \(1/\epsilon\) is parameterization by this gap size. If \(X\) is an optimization problem, then we define

**GAP-X**

**Input:** An instance \(x\) of \(X,\) an integer \(k,\) and a rational \(\epsilon > 0\) such that

\[
\begin{cases}
\text{either } \text{OPT}(x) \leq k \text{ or } \text{OPT}(x) > (1 + \epsilon)k & \text{if goal = min} \\
\text{either } \text{OPT}(x) \geq k \text{ or } \text{OPT}(x) < \frac{k}{1 + \epsilon} & \text{if goal = max}
\end{cases}
\]

**Parameter:** \([1/\epsilon]\)

**Decide:**

\[
\begin{cases}
\text{OPT}(x) \leq k, & \text{if goal = min,} \\
\text{OPT}(x) \geq k, & \text{if goal = max.}
\end{cases}
\]
An approximation scheme for optimization problem $X$ can be used to decide GAP-$X$. Let $X$ be a minimization problem (the argument for maximization problems is similar). If $(x, k, e)$ is a yes-instance of GAP-$X$, then $\text{OPT}(x) \leq k$, and the approximation scheme can produce a $(1 + \epsilon)$-approximate solution with cost at most $(1 + \epsilon)k$. On the other hand, if $(x, k, e)$ is a no-instance, then by assumption $\text{OPT}(x) > (1 + \epsilon)k$ holds, thus the cost of the solution produced by the approximation scheme is strictly greater than $(1 + \epsilon)k$. This means that deciding an instance $(x, k, e)$ of GAP-$X$ is not more difficult than obtaining a $(1 + \epsilon)$-approximate solution. Therefore, we have the following proposition.

**Proposition 6.2.** If optimization problem $X$ admits a PTAS, then GAP-$X$ is in XP. If $X$ admits an EPTAS, then GAP-$X$ is in FPT.

Recall that XP is the class of parameterized problems that can be decided in polynomial time for every fixed value of the parameter. Proposition 6.2 can be used to show that a problem $X$ does not have an EPTAS: if GAP-$X$ is $W[1]$-hard, then $X$ does not have an EPTAS unless FPT = $W[1]$. This technique is potentially more powerful than using Proposition 6.1: it might apply even to problems whose standard parameterizations are FPT, since $k$ is not a parameter of GAP-$X$.

To demonstrate the use of Proposition 6.2, we re-prove a result of [92] and show that PLANAR TMIN does not admit an EPTAS. The reduction is based on the framework of matrix-type reductions introduced in [84], but the presentation here is self-contained.

**Theorem 6.2.** GAP-PLANAR-TMIN is $W[1]$-hard. Thus PLANAR TMIN does not have an EPTAS, unless FPT = $W[1]$.

**Proof.** The reduction is from **Maximum Clique**. Given a graph $G$ and an integer $t$, we construct an instance of GAP-PLANAR-TMIN with $k = 2(t + 1)$ and $\epsilon = 1/(2k)$ such that

- if $G$ has a clique of size $t$, then the constructed instance has a solution with weight at most $k$;
- if $G$ does not have a clique of size $t$, then the constructed instance does not have a solution with weight at most $(1 + \epsilon)k$.

Assume that $G$ has $n$ vertices $\{1, 2, \ldots, n\}$. The constructed instance has $2n(t + 1)$ variables. The variables are arranged into blocks $A_{ij}$ $(1 \leq i \leq t, 0 \leq j \leq t)$ and $B_{ij}$ $(0 \leq i \leq t, 1 \leq j \leq t)$, where each block contains $n$ variables. The variables in block $A_{ij}$ (respectively, $B_{ij}$) will be denoted by $a_{i,j,s}$ (respectively, $b_{i,j,s}$) for $s = 1, \ldots, n$.

We construct $t^2$ DNFs such that $\phi_{ij}$ $(1 \leq i, j \leq t)$ contains variables only from blocks $A_{i-1,j}$, $A_{i,j}$, $B_{i-1,j}$, $B_{i,j}$. As shown in Fig. 1, the incidence graph is planar. The formulas are defined as

$$\phi_{ij} = \begin{cases} 
\bigvee_{s=1}^n (a_{i,j-1,s} \land a_{i,j,s} \land b_{i-1,j,s} \land b_{i,j,s}), & \text{if } i = j, \\
\bigvee_{(i,j) \in E(G)} (a_{i,j-1,s} \land a_{i,j,s} \land b_{i-1,j,s} \land b_{i,j,s}), & \text{otherwise}.
\end{cases}$$

This completes the description of the instance. The optimum is either at most $k$, or at least $k + 1 > (1 + \epsilon)k = (1 + 1/(2k))k = k + 1/2$. Therefore, the gap size requirement is satisfied.

Assume that $G$ has a clique $v_1, \ldots, v_r$. For every $i,j$, we set $a_{i,j,v_i}$ of block $A_{ij}$ and $b_{i,j,v_i}$ of block $B_{ij}$ to true. We claim that this assignment of weight $2(t + 1)$ satisfies every formula. Indeed, if $i = j = r$, then $\phi_{ij}$ is satisfied by the term $a_{i-1,j,v_i} \land a_{i,j,v_i} \land b_{i-1,j,v_i} \land b_{i,j,v_i}$; if $i \neq j$, then $\phi_{ij}$ is satisfied by the term $a_{i-1,j,v_i} \land a_{i,j,v_i} \land b_{i-1,j,v_i} \land b_{i,j,v_i}$ (note that $(v_i, v_j)$ is an edge of $G$, hence $\phi_{ij}$ has such a term). For the other direction, suppose that $G$ has no clique of size $t$, but there is a satisfying assignment of weight at most $(1 + \epsilon)k$. Notice that $(1 + \epsilon)k < k + 1$, hence the weight of the assignment is at most $k$. Furthermore, each block of variables has to contain at least one true variable, otherwise the adjacent formulas could not be satisfied. Therefore, the weight of the assignment is exactly $k$, and there is exactly one true variable in each block.

For every term of $\phi_{ij}$, there is an $s$ such that the term contains both $a_{i-1,j,s}$ and $a_{i,j,s}$. This means that if $\phi_{ij}$ is satisfied, then the true variables in blocks $A_{i-1,j}$ and $A_{i,j}$ have the same index. Furthermore, this is true for all the blocks in row $i$, hence there is a value $v_i$ such that $a_{i,j,v_i}$ is true for every $1 \leq j \leq t$. Similarly, for every column $j$, there is a value $v_j$ such that variable $b_{i,j,v_j}$ is true. We claim that $v_i = v_j$ for every $1 \leq i \leq t$. Indeed, if a term of $\phi_{ij}$ is satisfied, then both $a_{i,j}$ and $b_{i,j}$ have to be true for some $s$, which implies $s = v_i = v_j$. Now the vertices $v_1, \ldots, v_t$ form a clique:
\(v_i\) and \(v_j\) are not connected by an edge, then none of the terms in \(\phi_{i,j}\) are satisfied. This proves the correctness of the reduction. \(\square\)

### 6.3. Proving lower bounds

The fact that an optimization problem does not have an EPTAS (unless \(\text{FPT} = \text{W}[1]\)) does not rule out the possibility of having a PTAS of running time \(|x|^{O(k \log \log |1/\epsilon|)}\), for example. Such a running time would be almost as good as a polynomial-time algorithm. However, by a more careful analysis and by having a somewhat stronger complexity-theoretic assumption, we can actually prove lower bounds on the exponent of \(|x|\).

MAXIMUM CLIQUE can be solved in \(|x|^{O(k)}\) time by brute force. It is believed that this is essentially best possible; in fact, Chen et al. [93] show that an \(f(k) \cdot |x|^{O(k)}\) time algorithm would imply that 3SAT can be solved in \(2^{o(n)}\) time. The fastest known algorithm for \(n\)-variable 3-SAT runs in randomized time 1.32216\(^n\) [94] (improving on the \(2^n\) brute force algorithm). It is believed the running time of every algorithm must be exponential in \(n\), i.e. no algorithm with running time \(2^{o(n)}\) exists. This assumption is known as the Exponential Time Hypothesis (ETH) [95], and it is equivalent to the parameterized complexity conjecture \(\text{FPT} \neq \text{W}[1]\) (see [2, 96]).

This lower bound on MAXIMUM CLIQUE can be transferred to other parameterized problems via reductions. Assume that a problem \(Q\) is \(\text{W}[1]\)-hard by a parameterized reduction from MAXIMUM CLIQUE, and the reduction constructs instances such that the parameter is at most \(p(k)\) (for some function \(p\)), where \(k\) is the parameter of the maximum clique instance. (The definition of parameterized reduction requires that such a function \(p(k)\) exists.) If \(Q\) can be solved in time \(f(k) \cdot |x|^{O(k)}\), then the reduction implies that MAXIMUM CLIQUE can be solved in time \(f'(k) \cdot |x|^{O(p(k))}\). Therefore, the problem \(Q\) cannot be solved in time \(f'(k) \cdot |x|^{(p-1)(k)}\) for any function \(f'(k)\), unless ETH fails.

Proposition 6.1 shows that if an optimization problem admits an approximation scheme with running time \(f(1/\epsilon) \cdot |x|^{O(1/\epsilon)}\), then the standard parameterization can be solved in \(f(2k) \cdot |x|^{O(2k)}\) time. Therefore, a lower bound on the standard parameterization can be used to bound the efficiency of any PTAS for the problem. The \(\text{W}[1]\)-hardness results in Table 2 are obtained by a parameterized reduction from MAXIMUM CLIQUE, and these reductions increase the parameter at most by some polynomial function \(p(k)\). Therefore, assuming ETH, for each of these problems there is an integer \(c\) such that there is no \(f(k) \cdot |x|^{O(k^{1/\epsilon})}\) time algorithm for the standard parameterization, which implies that there is no \(f(1/\epsilon) \cdot |x|^{O((1/\epsilon)^{1/c})}\) time PTAS. The only exception is CLOSEST SUBSTRING: the reduction presented in [88] increases the parameter exponentially, thus all we can prove is that there is no \(f(1/\epsilon) \cdot |x|^{O(1/\epsilon^{1/4})}\) time PTAS.

In a similar way, we can obtain lower bounds using Proposition 6.2. For example, the reduction in Theorem 6.2 constructs instances where the parameter \(1/\epsilon\) is \(O(t^2)\). Therefore, we have the following corollary.

**Corollary 6.2.** There is no \(f(1/\epsilon) \cdot |x|^{O(1/\epsilon^{1/4})}\) time PTAS for \(\text{PLANAR Tmin}\), unless ETH fails.

This result is somewhat better than the \(f(1/\epsilon) \cdot |x|^{O(1/\epsilon^{1/2})}\) bound that follows from the proof in [92]. However, there is still a gap between this lower bound and the \(|x|^{O(1/\epsilon)}\) upper bound given by Khanna and Motwani [91]. It would be interesting to close this gap, possibly by proving stronger lower bounds. Using bidimensionality theory [97, 98], one may show that the standard parameterization of \(\text{PLANAR Tmin}\) can be solved in time \(f(k) \cdot |x|^{O(k^{1/4})}\). This means that no bound better than \(f(1/\epsilon) \cdot |x|^{O(1/\epsilon^{1/4})}\) can be obtained for \(\text{PLANAR Tmin}\) using Proposition 6.1. Note that if we are using Proposition 6.2, then there is no such apparent barrier. It seems that we have to abandon Proposition 6.1 and directly parameterize by \(1/\epsilon\), if we want to prove tight lower bounds on the efficiency of approximation schemes.

### 7. Conclusions

We have investigated several ways in which parameterized complexity and the theory of approximation algorithms can benefit from each other. It is not clear at the moment which directions will lead to fruitful areas of research. It seems that there are many examples of FPT-approximability for parameterized instances (as discussed in Section 3), and it can be reasonably expected that many such results will follow. It is much less clear to what extent standard FPT-approximability (Section 4, 5) will be important. The relative lack of examples might be the sign that we are missing some fundamental method. Evaluating the efficiency of approximation schemes (Section 6) has become a standard technique, with results of this type steadily accumulating. Besides obtaining such results for a wider range of problems, here the challenge is to prove tight results.

There are some further connections between parameterized complexity and approximation algorithms that we have not touched upon. *Kernelization* of a parameterized problem is a polynomial-time preprocessing algorithm that constructs an equivalent instance such that the size of the new instance can be bounded by a function of the parameter. For example, Nemhauser and Trotter [99] present a preprocessing algorithm for the MINIMUM VERTEX COVER problem that constructs an instance with at most \(2k\) vertices (where \(k\) is the parameter of the new instance). In many cases, such a linear-size kernelization algorithm can be used to obtain a constant-factor approximation algorithm. Therefore, any
lower bound on the approximation ratio (obtained by the PCP theorems) immediately translates into a lower bound on the kernel size. (A different approach for obtaining kernelization lower bounds is proposed in [100].)

Another interesting connection appears in the use of the iterative compression technique. The main idea of iterative compression is that instead of solving the search problem ‘find a solution of size $k$,’ we solve the possibly easier compression problem ‘given a solution of size $k + 1$, find a solution of size $k$.’ It turns out that for certain minimization problems if we can solve the compression problem, then we can solve the search problem as well by iteratively solving larger and larger instances [31, 60, 101–104]. Furthermore, as observed in [104], if we can solve the compression problem and have a constant-factor approximation algorithm, then the approximate solution can be compressed, and there is no need for the iterative trick. In fact, an approximation algorithm with arbitrary performance ratio function (Section 5) would be sufficient for this application.

In counting problems, the task is not only to check if there is a solution, but we have to determine the total number of solutions to the problem. The counting problem can be hard even for those problems where the existence of a solution is polynomial-time decidable. For example, checking whether a graph has a perfect matching can be done in polynomial time, but Valiant [105] has shown that counting the number of perfect matching is complete for the complexity class $\#P$. A similar increase in complexity appears in the case of parameterized problems: finding a path of length $k$ is FPT [16, 106, 107], but counting the number of these paths is $\#W[1]$-hard [108]. Given the hardness of exact counting, it is very natural to investigate the possibility of approximate counting. In the parameterized complexity framework, Arvind and Raman [109] present an FPT-time randomized approximation scheme for counting the number of subgraphs isomorphic to a bounded tree width graph $H$ (thus, in particular, the algorithm can be used to approximate the number of paths of length $k$). Recently, Müller [110] proved a parameterized complexity analog of a classical result of Stockmeyer by showing that approximately counting the number of solutions for a problem in $\text{W}[P]$ can be done in randomized FPT-time with a $\text{W}[P]$ oracle.

Finally, to motivate further research, we propose four open questions, related to the four main issues discussed in the paper.

- Investigate the approximability of geometric problems, parameterized by the dimension.
- Find a convincing example where no FPT-time exact algorithm is known for a problem, but it has a standard FPT-approximation algorithm with constant performance ratio.
- Is there a standard FPT-approximation algorithm for MAXIMUM CLIQUE or MINIMUM DOMINATING SET with some performance ratio function?
- Prove tight lower bounds (assuming ETH) on the efficiency of the PTAS for some natural problem (MPSAT, TMIN, geometric problems, etc.)

ACKNOWLEDGEMENTS

I would like to thank Mike Fellows for asking me to write this survey, and for the many ideas, comments and references that were essential in compiling the material. I am grateful to Martin Grohe and Magdalena Grüber for their comments on the manuscript.

REFERENCES


