Improved Approximations for Hard Optimization Problems via Problem Instance Classification

Improved Approximations for Hard Optimization Problems via Problem Instance Classification*

Hans-Joachim Böckenhauer, Juraj Hromkovič, and Tobias Mömke

Department of Computer Science, ETH Zurich, Switzerland
{hjb,juraj.hromkovic,tobias.moemke}@inf.ethz.ch

Abstract. Under the usual complexity-theoretic assumptions like $\mathcal{P} \neq \mathcal{NP}$, many practically relevant optimization problems are provably hard to solve or even to approximate. But most of these hardness results are derived for worst-case scenarios, and it is in many cases not clear whether the actual problem instances arising in practical applications exhibit this worst-case behaviour. Thus, a recent branch of algorithmic research aims at a more fine-grained analysis of the hardness of optimization problems. The main idea behind this analysis is to find some parameter according to which one can classify the hardness of problem instances. This approach does not only lead to new hardness results, but can also be used to design improved approximation algorithms for practically relevant subclasses of problem instances.
In this paper, we survey several different approaches for such improved approximation results achieved by a fine-grained classification of problem instances.

1 Introduction

Understanding and classifying the hardness of computational problems is one of the most fundamental goals in computer science. The model of the Turing machine [44] together with the Church-Turing thesis gives us a precise formal model of the class of problems that can be algorithmically solved and thus provides a sharp border between algorithmically solvable and algorithmically unsolvable problems. But every-day experience of the 1960s showed that many problems that are algorithmically solvable in principle are intractable from a practical point of view since all known algorithms are much too time-consuming.

This observation lead to the development of the fundamental concept of computational complexity [31, 43]. But, in contrast to the theory of computability, no formal model yielding a sharp border between "practically" solvable and unsolvable problems has yet been identified.

Instead, the hardness of a problem is often analyzed in a worst-case scenario. But this approach might yield misleading results for some practical applications since, for many problems, only a very few instances of any given length are really

* This work was partially supported by SNF grant 200021-132510/1.

as hard as the worst-case lower bound suggests. To overcome these difficulties, sometimes an average-case analysis is used. But, besides usually being a very challenging task, the results of any average-case analysis heavily depend on the assumed probability distribution of problem instances. Since the distribution of the actual inputs in a practical application usually is not known, this method can often yield misleading results as well. Instead of these traditional approaches, one would like to measure whether a given single problem instance is hard to process. Unfortunately, a sound formal definition of this is unattainable since the solution for a single instance can always be precomputed and integrated into the algorithm, leading to a constant complexity in any case.

One possible way out of this dilemma is to partition the set of problem instances into infinitely many infinitely large classes according to their hardness. Proving membership in one of these classes then gives an *upper* bound on the resources needed to process a specific instance. In a more general scope, the partitioning enables a fine-grained analysis of the hardness of a problem, where classes of tractable inputs can be identified.

One of the first approaches to establish such an infinite partitioning is called parameterized complexity. One identifies a suitable parameter k of the input and tries to bound the running time of the algorithm on an instance of length n by $\mathcal{O}(f(k) \cdot p(n))$ where f is an arbitrary computable function and p is some polynomial function. Intuitively speaking, such a bound on the time complexity means that the running time is exponential only in the parameter, but not in the input length. This parameterization of problem instances has proven to be a very successful technique, see, e. g., the books by Downey and Fellows [23] or Niedermeier [36] for an introduction. In this survey, we summarize some more recent approaches of partitioning the set of problem instances also for determining the hardness of computing *approximate* solutions for optimization problems.

There are several possibilities for directly extending the concept of parameterized complexity to optimization problems, we give a survey of known approaches in Section 3. In Section 4, we discuss the related concept of *stability* of approximation algorithms [7, 11]. Here, the idea is to allow only polynomial running time, but to determine the approximation ratio as a function depending on a parameter of the input. This parameter is typically determined by the distance of the considered problem instance from some set of easily solvable instances.

In Section 5, we survey two recent approaches of instance-classifying algorithm design. *Hybrid algorithms* find a partition of the problem instances such that, for each input, either the best known approximation or the best known running time of an exact algorithm can be significantly improved. For *win/win approximation algorithms*, we are able to prove that, for each input, the approximation ratio of at least one of two related problems can be improved. From another point of view, this means to parameterize the instances of one problem according to their approximability for the other problem.

2 Basic Definitions

In this section, we briefly recall the basic definitions of approximation algorithms. For a more detailed introduction to this topic, we refer to the books by Hromkovič [33], Ausiello et al. [4], or Vazirani [46].

An *optimization problem* U can be described by a quadruple $U = (L, \mathcal{M}, \mathit{cost}, \mathit{goal})$, where L denotes the set of problem instances, $\mathcal{M}(x)$ is the set of feasible solutions for each problem instance $x \in L$, *cost* is a function measuring the cost of any feasible solution for a given problem instance, and *goal* is the optimization goal, i. e., minimization or maximization. An algorithm A is called *consistent* for U if it computes a feasible solution $A(x) \in \mathcal{M}(x)$ for every problem instance $x \in L$.

Let A be a consistent algorithm for an optimization problem $U = (L, \mathcal{M}, \mathit{cost}, \mathit{goal})$. The *approximation ratio* $R_A(x)$ of A on the input x is defined as $R_A(x) = \max\{\mathit{cost}(A(x))/\mathit{Opt}_U(x), \mathit{Opt}_U(x)/\mathit{cost}(A(x))\}$, where $\mathit{Opt}_U(x)$ denotes the cost of an optimal solution for x. This definition ensures that the approximation ratio yields a value ≥ 1 for both minimization and maximization problems. For $\delta > 1$, we say that A is a *δ-approximation algorithm* for U if, for all $x \in L$, $R_A(x) \leq \delta$.

The definition can be generalized to describe also non-constant approximations by defining $R_A(n) = \max\{R_A(x) \mid x \text{ is input of size } n\}$. For every function $f\colon \mathbb{N} \to \mathbb{R}^+$, we call A an *$f(n)$-approximation algorithm* for U if $R_A(n) \leq f(n)$, for every $n \in \mathbb{N}$.

When dealing with approximations, we usually restrict our attention to optimization problems where (i) the problem instances are recognizable in polynomial time, (ii) the size of any feasible solution is polynomially bounded by the size of the input, (iii) the cost function is polynomial-time computable, and, (iv) for any solution candidate, it can be tested in polynomial time if it is a feasible solution. The class of problems satisfying these conditions is called $\mathcal{NPO}$.

The class $\mathcal{APX}$ is the subclass of problems from $\mathcal{NPO}$ for which there exists a δ-approximation algorithm, where $\delta \geq 1$ is a constant.

A consistent algorithm A for an optimization problem U is called a *polynomial-time approximation scheme (PTAS)* for U if, for every input pair $(x, \varepsilon) \in L \times \mathbb{R}^+$, A computes a feasible solution $A(x)$ with an approximation ratio of at most $1 + \varepsilon$ within a time that is polynomial in the size of x. If the running time can be bounded by a function that is polynomial both in the size of x and in ε^{-1}, we call A a *fully polynomial-time approximation scheme (FPTAS)*. The classes of problems from $\mathcal{NPO}$ admitting a PTAS or an FPTAS are called $\mathcal{PTAS}$ and $\mathcal{FPTAS}$, respectively.

3 Parameterized Approximation Algorithms

The idea behind parameterized algorithmics is the following: Consider some hard (e. g., $\mathcal{NP}$-hard) computing problem. If we want to design an exact algorithm solving the problem, we cannot expect a polynomial running time on all problem

instances, unless $\mathcal{P} = \mathcal{NP}$. Nevertheless, the complexity of an algorithm might vary a lot for different instances of the problem. Our goal now is to find a suitable parameterization of the set of inputs describing their hardness. In particular, we are interested in finding a parameter k for any input such that the super-polynomial part of the running time is tied to the value of the parameter only, but does not depend on the size of the input.

This concept was introduced by Downey and Fellows [21–23]. It can be formalized as follows: Let U be a computing problem and let L be the set of all problem instances of U. We call any polynomial-time computable function $\kappa\colon L \to \mathbb{N}$ a *parameterization* of U if, for infinitely many $k \in \mathbb{N}$, the set $\{x \in L \mid \kappa(x) = k\}$ is infinite. Intuitively speaking, a parameterization κ partitions the set of problem instances into infinitely many infinite subclasses. An algorithm A is called a *κ-parameterized polynomial-time algorithm* for U if A solves U and if there exists a polynomial function p and an arbitrary computable function f such that the running time of A on any problem instance x is bounded by $f(\kappa(x)) \cdot p(|x|)$, where $|x|$ denotes the size of x. If there exists a κ-parameterized algorithm for U, we say that the parameterized problem (U, κ) is *fixed-parameter tractable*. By $\mathcal{FPT}$ we denote the class of all fixed-parameter tractable parameterized problems. If the parameterization is clear from the context, we call a κ-parameterized polynomial-time algorithm simply an *fpt-algorithm*.

There are many different possibilities to choose such parameterizations. One possibility is to use some natural parameter revealing some structure of the input like the maximum vertex degree or the diameter in a graph, the maximum number of occurrences of a variable in a Boolean formula, or the alphabet size in a string. We call these *structural parameterizations* in the following.

Another possibility, which is often used in the existing literature for decision problems, is to consider the size of the solution as a parameter. We call this the *solution-size parameterization*, in the literature it can also be found under the name of *standard parametrization*. A well-known example is the *vertex cover problem*: Given an undirected graph and a natural number k, it asks whether there exists a set C of at most k vertices in the graph such that every edge is incident to at least one vertex from C. This problem, parameterized with the desired size of the vertex cover k as a parameter, is fixed-parameter tractable [21,23]. Similar results have been proven for many other parameterized decision problems, for an overview and a discussion of several design techniques for parameterized algorithms see, e. g., the books by Downey and Fellows [23] or Niedermeier [36].

On the other hand, many parameterized problems are not fixed-parameter tractable, unless $\mathcal{P} = \mathcal{NP}$. For proving such results, a full complexity theory has been developed, based on the concept of the so-called *$W[1]$-hardness*. See the books by Downey and Fellows [23] or Flum and Grohe [28] for an introduction to parameterized complexity theory.

While parameterized algorithms and parameterized complexity were mainly used for analyzing exact computations, there exist some approaches for extending

their applicability to optimization problems and approximation. We will give an overview of these approaches in this section, a further survey can be found in [35].

3.1 Efficient Polynomial-Time Approximation Schemes

Maybe the first attempt to connect parameterized algorithms and approximation was to look at approximation schemes from the point of view of parameterization. Only few optimization problems admit an FPTAS[1], and the running time of many known PTASs often is very high for any reasonable approximation ratio. Here, the concept of parameterization can sometimes help to establish a more fine-grained classification of PTASs. Remember that the input for a PTAS consists of an instance x of the corresponding optimization problem U together with some $\varepsilon > 0$ describing the desired approximation ratio $1 + \varepsilon$. We can view this as a parameterized problem by taking the desired approximation ratio as a parameter, i. e., by setting $\kappa(x, \varepsilon) = 1/\varepsilon$. A PTAS with a running time in $\mathcal{O}(f(1/\varepsilon) \cdot p(|x|))$ for some computable function f and some polynomial function p (or, in other words, an fpt-algorithm according to the parameter $1/\varepsilon$) is called an *efficient polynomial-time approximation scheme (EPTAS)*. This notion was introduced by Cesati and Trevisan [16].

One of the most prominent examples of an EPTAS is the one for the geometric traveling salesman problem given by Arora [3]. Other EPTASs were designed, for example, for scheduling on uniform processors [34], and for many so-called bidimensional problems on planar graphs, including feedback vertex set, vertex cover, minimum maximal matching, and a series of vertex-removal problems [19], see also [20] for an overview of bidimensionality. All of these algorithms are technically too involved to present the details in this survey. Instead, we illustrate the concept of EPTASs with a simple example taken from [9].

The *Steiner tree problem in graphs (STP)* is the following optimization problem: Given a complete edge-weighted graph $G = (V, E, c)$ and a subset $S \subseteq V$ of *terminal vertices*, the goal is to find a minimum-weight subgraph of G spanning all terminals. This problem is $\mathcal{APX}$-hard even in the case when all edge weights are taken from the set $\{1, 2, \ldots, r\}$ for some $r \in \mathbb{N}$, i. e., it does not admit any PTAS, unless $\mathcal{P} = \mathcal{NP}$ [6]. We now consider the following *reoptimization* variant[2] of this problem: Assume that an optimal solution Opt_{old} is given for an STP instance $(G = (V, E, c), S_{\text{old}})$, and we now want to compute a solution for a locally modified instance (G, S_{new}), where S_{new} is produced from S_{old} by adding a vertex to it. We call this problem, where the edge costs are again restricted to $\{1, 2, \ldots, r\}$, AddTerm-r-STP. It was shown in [9] that AddTerm-r-STP is $\mathcal{NP}$-hard.

Theorem 1 (Böckenhauer et al. [9]). *There exists an EPTAS for AddTerm-r-STP.*

[1] Only problems that are not strongly $\mathcal{NP}$-hard, i. e., that become polynomial-time solvable when the input is encoded in unary.

[2] For a motivation and a detailed introduction to the concept of reoptimization, see for instance [10].

Proof. Let $G = (V, E, c)$ be an edge-weighted graph, where $c\colon E \to \{1, \ldots, r\}$ for some constant $r \in \mathbb{N}$, let $S_{\text{old}}, S_{\text{new}} \subseteq V$ be two terminal sets such that $S_{\text{new}} = S_{\text{old}} \cup \{v\}$ for some $v \in V - S_{\text{old}}$, and let Opt_{old} be a minimum Steiner tree for (G, S_{old}). Let $1+\varepsilon$ be the desired approximation ratio, for some $\varepsilon > 0$. Then the following simple algorithm is an EPTAS for AddTerm-r-STP: Let $k := \lceil 1/\varepsilon \rceil$. If S_{new} contains at most $r \cdot k$ vertices, then compute an optimal solution using the Dreyfus-Wagner STP algorithm [26]. Otherwise, take the given old optimal solution and connect the new terminal v to it via an arbitrary edge.

We first analyze the approximation ratio in the second case of this algorithm. Since $|S_{\text{new}}| > k$ and every edge has cost at least 1, the cost of the optimal solution for the new instance is at least $r \cdot k$. Adding one edge to Opt_{old} costs at most r, i.e., the cost of the computed solution T_A can be estimated as $c(T_A) \leq c(Opt_{\text{old}}) + r$. Thus, the approximation ratio is

$$\frac{c(T_A)}{c(Opt_{\text{new}})} \leq \frac{c(Opt_{\text{old}}) + r}{c(Opt_{\text{new}})} \leq \frac{c(Opt_{\text{new}}) + r}{c(Opt_{\text{new}})} = 1 + \frac{r}{c(Opt_{\text{new}})} \leq 1 + \frac{r}{r \cdot k} \leq 1 + \varepsilon.$$

According to [26], the time complexity of calculating an optimal solution in the first case of the algorithm is in $O(n^2 \cdot 3^{r \cdot k})$. The time complexity of the remaining parts is negligible. Since r is a constant and $k = \lceil 1/\varepsilon \rceil$, all requirements for an EPTAS are satisfied. □

There are close relations between EPTASs and fpt-algorithms. In particular, the existence of an EPTAS is related to the fixed-parameter tractability according to the solution-size parameterization.

Theorem 2 (Cesati and Trevisan [16]). *If an optimization problem U admits an EPTAS, then the solution-size parameterization of U is in $\mathcal{FPT}$.* □

Corollary 1. *If the solution-size parameterization of an optimization problem U is $W[1]$-hard, then U does not admit an EPTAS, unless $\mathcal{P} = \mathcal{NP}$.* □

Corollary 1 directly implies that $W[1]$-hardness also rules out the possibility of an FPTAS. Thus, the theory of parameterized complexity can also be used to prove hardness results in classical approximation theory.

3.2 Structural Parameterizations

For using the approach of parameterization, many different structural parameters of the input can be used. For example, for graph problems, one can consider the maximum vertex degree, the diameter, the genus, the treewidth, etc. For satisfiability problems, the maximum length of clauses or the maximum number of occurrences of one variable are possible parameters.

Note that, strictly speaking, some parameters like the treewidth of a graph do not lead to valid parameterizations in the sense of the definition at the beginning of this section, since they are not computable in polynomial time. But there exists an fpt-algorithm (in the strict sense of the definition) for determining the

treewidth of a graph. Thus, it is meaningful to use it as a parameter as well; indeed, the treewidth parameterization is one of the most successful ones in parameterized algorithmics.

In this subsection, we give some examples where structural parameters help to yield improved approximations. We start with a problem which has been shown to be inapproximable as well as $W[1]$-hard, but for which we can design a simple constant-factor approximation algorithm with fpt-running time.

The *metric traveling salesman problem (Δ-TSP)* asks for finding a shortest Hamiltonian tour (i. e., a tour visiting each vertex exactly once) in a given edge-weighted complete graph, where the edge-weight function c satisfies the triangle inequality, i. e., for each three vertices u, v, w, we have $c(\{u, v\}) \leq c(\{u, w\}) + c(\{w, v\})$. In the *$\Delta$-TSP with deadlines ($\Delta$-DLTSP)*, we have additionally given a start vertex s and a subset of *deadline vertices* with prescribed deadlines which have to be visited by the tour before the cost of the tour exceeds these deadlines. In other words, the partial tour from s to any deadline vertex x with deadline $d(x)$ has to have a length of at most $d(x)$. It has been shown in [8] that the Δ-DLTSP is not approximable within a ratio of $((1 - \varepsilon)/2)|V|$, for any $0 < \varepsilon < 1$, unless $\mathcal{P} = \mathcal{NP}$, where V denotes the set of vertices in the input graph. The most natural parameterization for this problem is the number of deadline vertices. But, according to this parameterization, the problem is $W[1]$-hard since it was shown in [8] that it is $\mathcal{NP}$-hard even when restricted to instances with only two deadline vertices. Thus, neither approximation nor parameterized algorithms alone can help to solve this problem. But in the following we show that it is fruitful to combine both approaches.

For this, we first extend the notion of fixed-parameter tractability to approximation algorithms. A consistent algorithm A for a parameterized optimization problem (U, κ) is called a *δ-fpt-approximation algorithm* for (U, κ) if A computes an at most δ-approximative solution for every admissible input x for U with a running time bounded by $f(\kappa(x)) \cdot p(|x|)$ for some arbitrary computable function f and some polynomial function p.

Theorem 3 (Böckenhauer et al. [8]). *There exists a 2.5-fpt-approximation algorithm for Δ-DLTSP, parameterized by the number of deadline vertices.*

Proof. Consider a problem instance of Δ-DLTSP, consisting of a complete edge-weighted graph $G = (V, E, c)$ with metric edge-weight function c, a start vertex s, a set D of deadline vertices and a deadline function $d\colon D \to \mathbb{N}$ assigning a deadline to each deadline vertex. Let $k = |D|$ denote the number of deadline vertices.

We consider the following algorithm. It first computes a Hamiltonian tour H_C on all non-deadline vertices (but including the start vertex) using Christofides' algorithm [18]. Then it checks every linear order $\pi = (s, p_1, \ldots, p_k)$ on the vertices from $D \cup \{s\}$ and constructs the corresponding tour $H(\pi)$ visiting these vertices in this order. If this tour does not violate any deadline, the Christofides tour on the remaining vertices is appended at the end. The algorithm outputs the best of all tours constructed this way.

The running time of this algorithm can be estimated by $O(|V|^3 + k!k)$ since there are $k!$ possible orderings to be checked, the checking can be implemented to run in $O(k)$, and Christofides' algorithm has a running time in $O(|V|^3)$ [37]. Thus, this is an fpt-algorithm.

We now estimate the achieved approximation ratio. Consider an optimal solution *Opt* and the order π_{Opt} of the deadline in it. This order is checked by the algorithm, and the corresponding partial tour $H(\pi_{Opt})$ is no longer than *Opt* due to the triangle inequality. On the other hand, H_C is a 3/2-approximation on the subinstance induced by the non-deadline vertices since Christofides' algorithm is 3/2-approximative. Due to the triangle inequality, including more vertices into this tour cannot decrease the cost, and thus $c(H_C) + c(H(\pi_{Opt}) \leq 3/2 \cdot c(Opt) + c(Opt) = 2.5 \cdot c(Opt)$. □

We can extend the notion of fpt-approximation algorithms also to approximation schemes. An *fpt-AS* for a parameterized optimization problem (U, κ) is a consistent algorithm for U which outputs, for a given instance x of U and a given $\varepsilon > 0$, a $(1 + \varepsilon)$-approximate solution in time $f(\varepsilon, \kappa(x)) \cdot p(|x|)$ for some arbitrary computable function f and some polynomial p. This obviously is a generalization of the notion of an EPTAS because the super-polynomial part of the running time does not only depend on the desired approximation ratio, but also on the parameter $\kappa(x)$.

As an example for an fpt-AS, we mention an algorithm for the *partial vertex cover problem* which was presented in [35]. This is the following optimization problem: Given an undirected graph G and an integer k, the goal is to cover as many edges as possible with some subset of k vertices. This problem was shown to be $W[1]$-hard with respect to the parameter k in [30] and to be $\mathcal{APX}$-hard in [38].

Theorem 4 (Marx [35]). *The partial vertex cover problem admits an fpt-AS with respect to the parameterization* $\kappa(G, k) = k$. □

3.3 Solution-Size Parameterizations

The solution-size parameterization is the most studied type of parameterization for decision problems which seems to be quite natural in many cases and is also related to the existence of EPTASs as already mentioned in Subsection 3.1. But it is not trivial to extend this parameterization to optimization problems. The reason is that, for a hard optimization problem, determining the size of the solution is in most cases as hard as finding the optimal solution itself, but we expect a parameter to be easily computable from the given input.

For overcoming this difficulty, three different, but quite similar approaches were independently proposed in [15, 17, 24]. The main idea here is to add a parameter to the input describing a range of solution sizes for which the algorithm is expected to compute a good approximative solution. We present the definition from [17] here.

Let U be a minimization problem. A *solution-size δ-fpt-approximation algorithm* for U is a consistent algorithm for U that, given an input x for U

and some $k \in \mathbb{N}$ such that the value of the optimal solution $Opt(x)$ satifies $c(Opt(x)) \leq k$, computes a δ-approximate solution in time $O(f(k) \cdot p(|x|))$ for some computable function f and some polynomial function p. For pairs (x, k) such that $c(Opt(x)) > k$, the output of the algorithm can be arbitrary. The definition can be extended to maximization problems in an analogous way.

Not many applications of this concept are known, but some fpt-inapproximability results for the dominating set problem can be found in [25].

4 Stable Approximation Algorithms

In contrast to the parameterized approximation algorithms as described in the preceding section, for *stable* approximation algorithms the parameter is used for measuring the approximation ratio only, maintaining polynomial running time for every problem instance. The main idea is that, for many hard-to-approximate optimization problems, there exists a subset of relatively easily approximable instances, and all other instances can be partitioned into infinitely many classes according to their distance to this easy kernel of the problem, with respect to some appropriate distance measure. Now the goal is to find an algorithm that is consistent for all instances and achieves good approximations for the easy kernel, such that the achieved approximation ratio depends on the distance from the kernel only, but not on the size of the input. This concept of approximation stability was introduced in [7], a detailed survey can be found, e. g., in [11].

More formally, approximation stability can be defined as follows. We start with defining how to measure the distance between problem instances. Let $\overline{U} = (L, \mathcal{M}, cost, goal)$ be an optimization problem and let $U = (L_I, \mathcal{M}, cost, goal)$ be a subproblem of $\overline{U}$, where $L_I \subsetneq L$. Any function $h_L \colon L \to \mathbb{R}^+$ satisfying $h_L(x) = 0$ for all $x \in L_I$ is called a *distance function for U according to L_I*. For any $r \in \mathbb{R}^+$, we define $Ball_{r,h_L}(L_I) = \{w \in L \mid h_L(w) \leq r\}$ to be the set of instances from L which are at distance at most r from L_I.

This definition now enables us to formally define stable approximation algorithms. We consider an ε-approximation algorithm A for U for some $\varepsilon \in \mathbb{R}^+$ which is consistent for $\overline{U}$ and some $p \in \mathbb{R}^+$. We say that A is *p-stable according to h_L* if, for every real number $0 \leq r \leq p$, there exists some $\delta_{r,\varepsilon} \in \mathbb{R}^+$ such that A is a $\delta_{r,\varepsilon}$-approximation algorithm on the subproblem of $\overline{U}$ restricted to the instances in $Ball_{r,h_L}(L_I)$. The algorithm A is called *stable according to h_L* if it is p-stable according to h_L for every $p \in \mathbb{R}^+$. If there exists a $p > 0$ such that A is not p-stable, then A is called *unstable*.

As an example, let us consider the traveling salesman problem (TSP). For general edge weights, the TSP is not approximable within $p(n)$ for any polynomial function p, where n denotes the number of vertices in the input graph. On the other hand, if we restrict the problem to edge-weighted graphs satisfying the triangle inequality, the resulting subproblem Δ-TSP admits a 1.5-approximation due to Christofides' algorithm [18]. The idea now is to take the Δ-TSP as the easy problem kernel and to define some distance to metricity for all other TSP inputs. This can be done by considering a *relaxed triangle*

inequality. We say that an edge-weighted graph $G = (V, E, c)$ satisfies the β-triangle inequality for some $\beta \geq 1$ if, for all vertices $u, v, w \in V$, the inequality $c(\{u, v\}) \leq \beta \cdot (c(u, w) + c(w, v))$ holds. Now we can define a distance function h_{TSP} measuring the distance from Δ-TSP for every TSP instance G by $h_{\text{TSP}}(G) = \beta_G - 1$, where $\beta_G \geq 1$ is the minimum value for β such that G satisfies the β-triangle inequality.

It has been shown in [7] that Christofides' algorithm is unstable for h_{TSP}, but a modification of the algorithm can be shown to be stable. Other stable TSP algorithms were developed in [1, 2, 5]; for an overview, see [11].

Theorem 5 (Böckenhauer et al. [7]). *There exists a stable approximation for TSP achieving an approximation ratio of* $1.5 \cdot \beta^2$ *on graphs satisfying the* β*-triangle inequality.* □

5 Hybrid Algorithms and Win/Win Approximations

In the previous sections, we have discussed several ways to combine techniques from parameterization and approximation. Another intriguing approach is to use a single parameter for both exact computations and approximations such that, for any value of the parameter, we obtain either an improved exact algorithm or an improved approximation. Algorithms achieving this goal belong to the class of so-called hybrid algorithms which originate from the *algorithm selection problem* [40], the problem to select an appropriate algorithm for an input according to some given selection criteria. A *hybrid algorithm* consists of a collection of algorithms for a problem U and a *selector* S that decides which algorithm from the collection is used for a given input. In the context of this survey, we are interested in those hybrid algorithms that, for a set M of complexity measures, guarantee to compute a good result for any given input with respect to at least one measure $m \in M$.

The set of complexity measures can contain, for example, the worst-case running time and the approximation ratio achievable in polynomial time. An example of a selector S is to identify a parameter of the problem instance and to decide accordingly whether to compute an exact solution or an approximation. The goal is that, according to the value of the parameter, we can either ensure an improved running time or an improved approximation ratio. More precisely, in the case of exact computations, the running time is parameterized with respect to the identified parameter and improves the best known fpt-algorithm for the problem. In the case of approximation, we can ensure that the approximation ratio improves over the best known worst-case approximation ratio, depending on the parameter, similarly as in the case of stable approximation algorithms.

In the following, we give an example of a hybrid algorithm for the maximum cut problem in unweighted graphs (MaxCut). Measured in the number of edges m, the currently best exact algorithm solves MaxCut in time $O(2^{m/5})$ [42]. The worst-case approximation ratio is about 1.1383 using extensive computational power [29]; the best algorithm that is not based on semidefinite programming is 2-approximative and runs in time $O(n+m)$, where n is the number of vertices [41].

The notation O^* denotes an asymptotic upper bound where polynomial factors are omitted.

Theorem 6 (Vassilevska et al. [45]). *For any $\varepsilon > 0$, there is a hybrid algorithm for MaxCut that either computes an exact solution in time $O^*(2^{\varepsilon m})$ or an expected $(4/(2+\varepsilon))$-approximation in linear time.*

Proof. Given a graph $G = (V, E)$ with $|V| = n$ vertices and $|E| = m$ edges, the hybrid algorithm computes a maximal matching M in linear time, i. e., a matching that contains at least one vertex of every edge from E. Now, the selector decides whether to compute an optimal solution or an approximation depending on the size of M. If $|M| < \varepsilon m/2$, the algorithm computes an exact solution as follows. Let V_M be the set of vertices in M. For each partition of V_M into two sets, the algorithm distributes the vertices from $G \backslash M$ greedily, i. e., the vertices from $G \backslash M$ are distributed one after the other such that each time the cut is maximized. Then it takes the best solution obtained this way. It is clear that this algorithm runs in time $O^*(2^{\varepsilon m})$, since there are $2^{\varepsilon m}$ possible partitions of $\varepsilon m > |V_M|$ vertices. To show that the outcome is an optimal solution, note that at least one of the tested partitions, say (M_1, M_2), coincides with the partition defined by an optimal solution. Since M is a maximal matching, the vertices in $G \backslash M$ form an independent set. Therefore, the optimal distribution of those vertices depends only on M_1 and M_2 and thus greedily distributing the vertices is sufficient.

The remaining case is that $|M| \geq \varepsilon m/2$ and the selector chooses to compute an approximate solution. In this case, the algorithm separately distributes the vertices from M and those from $G \backslash M$ into two sets M_1 and M_2. For each edge $\{u, v\}$ in M, with probability $1/2$ it puts u into M_1 and v into M_2 and with probability $1/2$ it puts u into M_2 and v into M_1. This way, u and v are not in the same set. The algorithm puts each of the remaining vertices with probability $1/2$ to M_1 and otherwise to M_2. Now, the expectation is that half of the edges from E not contained in M are in the cut. Additionally, all $\varepsilon m/2$ edges from M are in the cut. Therefore, the expectation of the total number of edges in the cut is $\varepsilon m/2 + (m - \varepsilon m/2)/2 = m/2 + \varepsilon m/4$.

Since the value of an optimal solution can be trivially bounded from above by m, this results in the approximation ratio $Opt/(m/2 + \varepsilon m/4) \leq m/((2m + \varepsilon m)/4) = 4/(2+\varepsilon)$. □

For $\varepsilon < 1/5$, the algorithm from Theorem 6 either improves over the best known exact algorithm or over the best known linear-time approximation algorithm.

We continue with a related concept, the so-called *win/win algorithms*. A win/win algorithm computes — similar to a hybrid algorithm — more than one solution. In contrast to hybrid algorithms, however, win/win algorithms only use a single complexity measure but for a collection of different problems that share the same set of input instances. In other words, given one instance for two problems, a win/win algorithms guarantees to deliver a good solution for at least one of the problems. The win/win approach originates from parameterized computations and is used there as a technique for kernelization. *Kernelization* is

a preprocessing that, for a parameterized problem with parameter k, transforms a given problem instance into a new one such that the size of the transformed instance only depends on k. Then it suffices to handle the transformed input instead of the original one. In other words, a kernelization ensures us that we can either directly decide the parameterized problem in polynomial time or we can perform the computations on a smaller instance (possibly with a smaller parameter). An important result in parameterized complexity is that a problem is in $\mathcal{FPT}$ if and only if it has a kernelization. An overview on kernelization techniques can be found, e. g., in [36].

We now sketch an example from Prieto and Sloper [39] that shows how to use win/win algorithms for kernelization. The *k-vertex cover problem* (k-VC) is to decide whether there is a set of k vertices within a graph $G = (V, E)$ such that each edge $e \in E$ is incident to at least one of the vertices. The win/win approach shows that this problem is closely related to the problem of finding a spanning tree in G that has at least k internal vertices (k-IST).

The relation is based on finding a spanning tree in G such that either its leaves form an independent set in G or there are only two leaves. Note that, if there are only two leaves, then the tree is a Hamiltonian path and thus an $(n-2)$-internal spanning tree in a graph with n vertices.

Lemma 1 (Prieto and Sloper [39]). *Given a graph G, one can find in polynomial time a spanning tree T in G such that either T is a Hamiltonian path or the leaves of T form an independent set.*

Proof. The main ingredient of the proof is to compute a spanning tree T' in G and to transform T' into another spanning tree T that is either a Hamiltonian path or a spanning tree, where the leaves form an independent set. The transformation is done by successively searching for pairs of leaves u, v that are connected in G. By adding the edge $\{u, v\}$ to the spanning tree and removing an edge with at least three incident neighbors from the unique path connecting u and v, we obtain a new spanning tree, where the number of leaves is reduced by at least one. This way, the transformation ends with a tree T such that either no vertex in T has a degree higher than two (i. e., T is a Hamiltonian path) or there is no edge between leaves of T (i. e., the leaves form an independent set). □

If the leaves form an independent set, then the internal vertices form a vertex cover. Therefore, using Lemma 1, we obtain the following win/win result.

Theorem 7 (Prieto and Sloper [39]). *There is a polynomial-time algorithm that, for a given graph G, either computes a vertex cover of at most k vertices or a spanning tree with at least k internal vertices.* □

The stated result is interesting by its own, but additionally one can use it to obtain a kernelization of k-IST.

Theorem 8 (Prieto and Sloper [39]). *For any graph G, one can either find a k-internal spanning tree T in polynomial time or G can be transformed in polynomial time to a graph G' with at most $2k^3 + k^2 + 2k$ vertices such that G has a k-internal spanning tree if and only G' has.* □

The concept of win/win algorithms smoothly translates to approximation. As in win/win algorithms in parameterized computations, we are given one input that is a valid instance of two different problems. But instead of searching for an exact solution, we want to find approximate solutions for these problems. We aim for algorithms that guarantee at least one of the solutions to be improved compared to its known upper bounds on the worst-case approximation ratio. We can see the win/win approximation as a type of parameterization, where the approximation ratio achieved by one of the problems is the parameter for the other one.

An application of Lemma 1, presented in [27], relates the traveling salesman problem with edge costs restricted to 1 and 2, $(1,2)$-TSP, to the independent set problem (IS). Given a set of edges E, let E_1 be E restricted to the edges of cost 1. The following lemma is a slightly strengthened version of a result from [27].

Lemma 2. *Given a complete weighted graph $G = (V, E, c)$ with edge costs restricted to 1 and 2, one can compute a Hamiltonian path of length L in G and an independent set of size I in the unweighted graph $G' = (V, E_1)$ such that $L - I \leq n - 2$, where $n = |V|$.*

Proof. We apply Lemma 1 to each connected component of G'. Then I is the size of a maximal independent set composed of the leaves of all computed spanning trees within all components. (Note that each such maximal independent set has the same size: It includes all leaves except some of those that are part of a Hamiltonian path within a component.) In particular, each component contributes at least one vertex counted in I.

In order to form a Hamiltonian path of length L, we handle the components of G' separately. For each component C, let T_C be the spanning tree computed for C. We form a path P_C in G by connecting the vertices of C in the order of a depth-first search in T_C starting from a leaf. Note that all edges of P_C that are not in T_C contain at least one leaf of T_C. Thus, we can assign each edge of cost 2 to a leaf of T_C. The leaf from which the depth-first search starts is not assigned to any edge of cost 2. Therefore, for each component C, the number of vertices counted in the independent set I is at least the number of edges of cost 2 in P_C plus one. Finally, we have to connect all paths P_C to one Hamiltonian path in G using edges of cost 2. The claim of the lemma follows immediately, since the number of such edges is the number of components minus one. □

With this preparation, the actual win/win result is not hard to obtain.

Theorem 9 (Eppstein [27]). *Given a complete weighted graph G with edge weights 1 or 2, there is a win/win approximation algorithm that, for any $\varepsilon > 0$, either computes a $(1+\varepsilon)$-approximative $(1,2)$-TSP tour or a $(1/\varepsilon)$-approximative independent set in G', where G' is defined as in Lemma 2.*

Proof. Let L^* be the length of an optimal Hamiltonian tour in G and let I^* be the size of a maximum independent set of G'. If $L \leq (1+\varepsilon)(n-1)$, then we are done, because we are sure that $L^* \geq n - 1$. Otherwise, note that $\varepsilon \leq 1$ holds. Using Lemma 2, we get $I \geq L - n + 2 > (1+\varepsilon)(n-1) - (n-1) + 1 \geq \varepsilon n$.

Since $I^* \leq n$, the approximation ratio for the maximum independent set is $I^*/I \leq n/(\varepsilon n) = 1/\varepsilon$. □

The idea of Theorem 9 was used as a technique to find a PTAS for a graph embedding problem in [14].

For some pairs of problems, one can show that, unless $\mathcal{P} = \mathcal{NP}$, there are no win/win algorithms that lead to improved approximation ratios.

Theorem 10 (Eppstein, [27]). *For all $\varepsilon > 0$, unless $\mathcal{P} = \mathcal{NP}$, there is no polynomial-time win/win approximation algorithm for IS and the maximum clique problem that achieves an approximation ratio of $n^{1-\varepsilon}$ for either of the problems, where n is the number of vertices.* □

The pairs of problems considered above for the positive results were related in a way that has similarities with duality in linear programming in the sense that they rely on the interaction of minimization and maximization. To show that this is not a specific property of win/win, we present a win/win approximation from [12] that relates two minimization problems, namely the metric Hamiltonian path problem with prespecified start and end vertex (Δ-HPP$_2$) and the Δ-TSP.

Currently, the algorithm of Christofides [18] has the best proven approximation ratio for the Δ-TSP, which is 1.5. A slight modification of that algorithm was shown by Hoogeveen [32] to be 5/3-approximative for the Δ-HPP$_2$. The two problems are strongly related in the sense that, for any given metric graph, either we are guaranteed to obtain an approximation ratio for Δ-TSP that is significantly better than 1.5-approximative or we can solve the Δ-HPP$_2$ better than 5/3-approximatively for any choice of end-vertices.

To this end, let us consider an algorithm A that works as follows. The input of A is a complete edge-weighted metric graph $G = (V, E, c)$ and two vertices u and v. Then A runs both Christofides' algorithm and Hoogeveen's algorithm on the input using the same minimum-cost spanning tree.

Let Opt_C and Opt_P denote the optimal solutions for the Δ-TSP and the Δ-HPP$_2$ given an input G, u, v and let H_C and H_P be the solutions computed by A. Then we define $\alpha := c(H_P)/c(Opt_P)$ to be the approximation ratio of the computed Hamiltonian path and $\beta := c(H_C)/c(Opt_C)$ to be the approximation ratio of the computed Hamiltonian tour.

Theorem 11 (Böckenhauer et al. [12]). *The approximation ratio α is at most $1+1.5/(2\beta)$, independent of the choice of u and v. In particular, only pairs of approximation ratios from or below the shaded area in Figure 1 are possible.* □

6 Conclusion

This paper provides a survey of some currently used ways of classifying instances of hard problems with respect to their computational hardness. On the one hand, this kind of effort shows why some hard problems can be successfully solved in practice. On the other hand, the presented approaches help to understand the

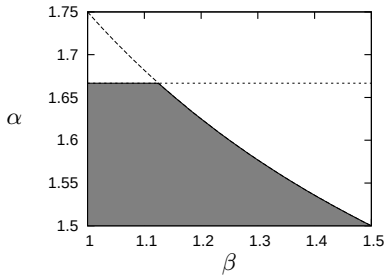

Fig. 1. Upper bound on the approximation ratio from Theorem 11. The horizontal line displays the approximation ratio $\alpha \leq 5/3$ proven in [32]

nature of the hardness of some algorithmic problems by specifying the properties of the instances that lie in the hardest kernels of the considered problems.

References

1. Andreae, T.: On the traveling salesman problem restricted to inputs satisfying a relaxed triangle inequality. Networks 38(2), 59–67 (2001)
2. Andreae, T., Bandelt, H.J.: Performance guarantees for approximation algorithms depending on parameterized triangle inequalities. SIAM Journal on Discrete Mathematics 8, 1–16 (1995)
3. Arora, S.: Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. Journal of the ACM 45(5), 753–782 (1998)
4. Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., Protasi, M.: Complexity and Approximation. Springer-Verlag, Berlin (1999)
5. Bender, M., Chekuri, C.: Performance guarantees for TSP with a parametrized triangle inequality. Information Processing Letters 73, 17–21 (2000)
6. Bern, M.W., Plassmann, P.E.: The Steiner problem with edge lengths 1 and 2. Information Processing Letters 32(4), 171–176 (1989)
7. Böckenhauer, H.-J., Hromkovič, J., Klasing, R., Seibert, S., Unger, W.: Towards the notion of stability of approximation for hard optimization tasks and the traveling salesman problem. In: Bongiovanni, G.C., Gambosi, G., Petreschi, R. (eds.) Proc. of the 4th Italian Conference on Algorithms and Complexity (CIAC 2000). Lecture Notes in Computer Science, vol. 1767, pp. 72–86. Springer-Verlag, Berlin (2000)
8. Böckenhauer, H.-J., Hromkovič, J., Kneis, J., Kupke, J.: The parameterized approximability of TSP with deadlines. Theory of Computing Systems 41(3), 431–444 (2007)
9. Böckenhauer, H.-J., Hromkovič, J., Královič, R., Mömke, T., Rossmanith, P.: Reoptimization of Steiner trees: Changing the terminal set. Theoretical Computer Science 410(36), 3428–3435 (2009)
10. Böckenhauer, H.-J., Hromkovič, J., Mömke, T., Widmayer, P.: On the hardness of reoptimization. In: Geffert, V., Karhumäki, J., Bertoni, A., Preneel, B., Návrat, P., Bieliková, M. (eds.) Proc. of the 34th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2008). Lecture Notes in Computer Science, vol. 4910, pp. 50–65. Springer-Verlag, Berlin (2008)

11. Böckenhauer, H.-J., Hromkovič, J., Seibert, S.: Stability of approximation. In: Gonzalez, T.F. (ed.) Handbook of Approximation Algorithms and Metaheuristics, Chapter 31. Chapman & Hall/CRC (2007)
12. Böckenhauer, H.-J., Klasing, R., Mömke, T., Steinová, M.: Improved approximations for TSP with simple precedence constraints. In: Calamoneri, T., Díaz, J. (eds.) Proc. of the 7th International Conference on Algorithms and Complexity (CIAC 2010). Lecture Notes in Computer Science, vol. 6078, pp. 61–72. Springer-Verlag (2010)
13. Bodlaender, H.L., Langston, M.A. (eds.): Parameterized and Exact Computation, Second International Workshop, IWPEC 2006, Zürich, Switzerland, September 13-15, 2006, Proceedings, Lecture Notes in Computer Science, vol. 4169. Springer-Verlag (2006)
14. Cabello, S., Eppstein, D., Klavžar, S.: The Fibonacci dimension of a graph. CoRR abs/0903.2507 (2009)
15. Cai, L., Huang, X.: Fixed-parameter approximation: Conceptual framework and approximability results. Algorithmica 57(2), 398–412 (2010)
16. Cesati, M., Trevisan, L.: On the efficiency of polynomial time approximation schemes. Information Processing Letters 64(4), 165–171 (1997)
17. Chen, Y., Grohe, M., Grüber, M.: On parameterized approximability. In: Bodlaender and Langston(eds.): Proc. of the Second International Workshop on Parameterized and Exact Computation (IWPEC 2006). Lecture Notes in Computer Science, vol. 4169, pp. 109–120. Springer-Verlag (2006)
18. Christofides, N.: Worst-case analysis of a new heuristic for the travelling salesman problem. Tech. Rep. 388, Graduate School of Industrial Administration, Carnegie-Mellon University (1976)
19. Demaine, E.D., Hajiaghayi, M.T.: Bidimensionality: New connections between FPT algorithms and PTASs. In: Proc. of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2005). pp. 590–601. Society for Industrial and Applied Mathematics (2005)
20. Demaine, E.D., Hajiaghayi, M.: The bidimensionality theory and its algorithmic applications. Computer Journal 51(3), 292–302 (2008)
21. Downey, R.G., Fellows, M.R.: Fixed-parameter tractability and completeness i: Basic results. SIAM Journal on Computing 24(4), 873–921 (1995)
22. Downey, R.G., Fellows, M.R.: Fixed-parameter tractability and completeness ii: On completeness for $W[1]$. Theoretical Computer Science 141, 109–131 (1995)
23. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Monographs in Computer Science, Springer-Verlag, New York (1999)
24. Downey, R.G., Fellows, M.R., McCartin, C.: Parameterized approximation problems. In: Bodlaender and Langston(eds.): Proc. of the Second International Workshop on Parameterized and Exact Computation (IWPEC 2006). Lecture Notes in Computer Science, vol. 4169, pp. 121–129. Springer-Verlag (2006)
25. Downey, R.G., Fellows, M.R., McCartin, C., Rosamond, F.A.: Parameterized approximation of dominating set problems. Information Processing Letters 109(1), 68–70 (2008)
26. Dreyfus, S.E., Wagner, R.A.: The Steiner problem in graphs. Networks 1, 195–207 (1971/72)
27. Eppstein, D.: Paired approximation problems and incompatible inapproximabilities. In: Charikar, M. (ed.) Proc. of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2010). pp. 1076–1086. Society for Industrial and Applied Mathematics (2010)

28. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer-Verlag (2006)
29. Goemans, M.X., Williamson, D.P.: Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. Journal of the ACM 42(6), 1115–1145 (1995)
30. Guo, J., Niedermeier, R., Wernicke, S.: Parameterized complexity of vertex cover variants. Theory of Computing Systems 41(3), 501–520 (2007)
31. Hartmanis, J., Stearns, R.E.: On the computational complexity of algorithms. Transactions of the American Mathematical Society 117, 285–306 (1965)
32. Hoogeveen, J.A.: Analysis of Christofides' heuristic: some paths are more difficult than cycles. Operations Research Letters 10(5), 291–295 (1991)
33. Hromkovič, J.: Algorithmics for Hard Problems. Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics. Springer-Verlag, Berlin (2003)
34. Jansen, K.: An EPTAS for scheduling jobs on uniform processors: Using an MILP relaxation with a constant number of integral variables. SIAM Journal on Discrete Mathematics 24(2), 457–485 (2010)
35. Marx, D.: Parameterized complexity and approximation algorithms. The Computer Journal 51(1), 60–78 (2008)
36. Niedermeier, R.: Invitation to Fixed Parameter Algorithms (Oxford Lecture Series in Mathematics and Its Applications). Oxford University Press, USA (March 2006)
37. Papadimitriou, C.H., Steiglitz, K.: Combinatorial Optimization: Algorithms and Complexity. Prentice-Hall (1982)
38. Petrank, E.: The hardness of approximation: gap location. Computational Complexity 4, 133–157 (1994)
39. Prieto, E., Sloper, C.: Either/or: using vertex cover structure in designing FPT-algorithms—the case of k-Internal Spanning Tree. In: Dehne, F.K.H.A., Sack, J.R., Smid, M.H.M. (eds.) Proc. of the 8th International Workshop on Algorithms and Data Structures (WADS 2003). Lecture Notes in Computer Science, vol. 2748, pp. 474–483. Springer-Verlag, Berlin (2003)
40. Rice, J.R.: The algorithm selection problem. Advances in Computers 15, 65–118 (1976)
41. Sahni, S., Gonzalez, T.F.: P-complete approximation problems. Journal of the ACM 23(3), 555–565 (1976)
42. Scott, A.D., Sorkin, G.B.: Faster algorithms for MAX CUT and MAX CSP, with polynomial expected time for sparse instances. In: Arora, S., Jansen, K., Rolim, J.D.P., Sahai, A. (eds.) Proc. of the 7th International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM 2003). Lecture Notes in Computer Science, vol. 2764, pp. 382–395. Springer-Verlag (2003)
43. Stearns, R.E., Hartmanis, J., II, P.M.L.: Hierarchies of memory limited computations. In: Proc. of the 6th Annual Symposium on Switching and Automata Theory. pp. 179–190. IEEE (1965)
44. Turing, A.M.: On computable numbers, with an application to the entscheidungsproblem. Proceedings of the London Mathematical Society 2(42), 230–265 (1936)
45. Vassilevska, V., Williams, R., Woo, S.L.M.: Confronting hardness using a hybrid approach. In: Proc. of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2006). pp. 1–10. Society for Industrial and Applied Mathematics, New York (2006)
46. Vazirani, V.V.: Approximation Algorithms. Springer-Verlag (2004)