Structural Properties of Hard Problem Instances

Tobias Mömke
KTH — Royal Institute of Technology
Stockholm, Sweden
moemke@kth.se

ABSTRACT

Hardness results for NP-hard problems are usually achieved for worst-case scenarios. In many cases, however, the worst-case behavior seems to be restricted to quite special cases and it may deviate considerably from the behavior for problem instances that are given as input. We address this effect by giving a survey of methods to achieve 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. In this spirit, we give an overview on parameterization, stability of approximation, a specific class of Hybrid algorithms, and win/win algorithms. We especially focus on the last topic which aims to relate different problems. For instance, we characterize instances for the metric TSP according to the solution computed by Hoogeveen's 5/3-approximation algorithm for the problem to find a Hamiltonian path with prespecified ends in the same metric graph. Our analysis reveals that the sets of the hardest instances of both problems for Christofides' and Hoogeveen's algorithm are disjoint in the sense that any instance is guaranteed to allow at least one of the two algorithms to achieve a significantly improved approximation ratio. In particular, we show that in any worst-case instance of Christofides' algorithm we can compute a 1.5-approximations for the Hamiltonian paths between any pairs of vertices.

1. INTRODUCTION

In this survey we want to address the discrepancy between the hardness of different problem instances of a hard problem. We want to analyze which structural properties of a problem instance improve the behavior of algorithms and, provided that a problem instance is hard, whether this hardness itself is a property that we can use.

In order to talk about hard problems, we first have to discuss the term *hardness*. Intuitively, we may consider a problem to be hard if the computational power that is needed to solve the problem exceeds the resources that are available for this task. This means that the term hardness is closely related to further conditions: a real-time application has much stricter time constraints than an algorithm that is used once to compute a good schedule for setting up a production line. These conditions, however, depend on the speed of currently available computers and they do not provide an exact specification of hardness.

Here, we want to talk about hardness on a more general level that is independent of contemporary technology. To this end, we need a more robust definition of hardness. The most popular definition that is in accordance with our requirements is that problems are "easy" if the runtime of any problem instance is asymptotically bounded by a polynomial and that they are "hard" if there is a problem instance that has a runtime that cannot be bounded by a polynomial. In other words, a hard problem is a problem that is hard in the worst case.

This definition offers a robust base for a well founded analysis of hardness. The drawback of this definition is that polynomial runtime does not guarantee that we can find solutions efficiently (this is the case if there is a factor with a large exponent) and — due to the use of worst-case instances in the definition — not every exponential-time algorithm runs unreasonably long. We consider the first point as a minor issue, since for most problems that are known to be solvable in polynomial time, eventually an efficient algorithm is found. The second point is much more severe. For instance, the simplex algorithm has exponential runtime in the worst case but in general it is considered to run fast [23].

In order to address this issue in a consistent way, we aim for a fine-grained analysis that considers the choice of problem instances that we specify as feasible input. In the context of this survey we restrict ourselves to decision problems in NP and optimization problems in NPO, that is, optimization problems that can be solved in polynomial time by a nondeterministic Turing machine. Additionally, for a problem to be in NPO, the feasibility of a solution is required to be verifiable deterministically in polynomial time. In order to talk about hardness within this complexity class, we have to assume that the $P \neq NP$ conjecture holds. We are interested in those problems from NPO that are NP-hard.

At this point, we have to define hardness of problem instances in a sound way. In particular, we cannot talk about the hardness of a single instances within NPO, since for each specific instance we can construct an efficient algorithm by simply providing one of its optimal solutions as additional information hard-coded in an algorithm.

Instead, we aim for a partition of the set of all problem instances into infinitely many classes each of which has infinitely many members. In other words, we aim to analyze given problem instances in order to find certain structural properties. Then we use these properties in order to classify the hardness of these problem instances. This way, determining the class of a given problem instance gives us an upper bound on its hardness.

The most famous approach that provides a classification as described above is parameterization. Section 2 gives an introduction to this topic and gives an overview on stability of approximation, which is closely related to parameterization but based on approximation instead of exact algorithms.

In Section 3 we focus on the relation between different complexity measures and, in Section 4, the relation between different problems. The idea is that we have several goals that we want to achieve, each of which is known to be hard in the worst case. While it is clear that we cannot guarantee to achieve all of the goals, we can guarantee that we can achieve some. This concept relates to parameterization and stability of approximation in the sense that now a computed solution becomes a property according to which we determine the hardness.

2. PARAMETERIZATION AND STABILITY OF APPROXIMATION

The main idea of parameterization is to confine the hardness of a problem to a parameter. Such a parameter can be, for instance, the size of the computed solution (this is also called the *standard parameterization*) or the number of variables in a boolean formula. More formally, given a decision problem U and a natural number k, the pair (U, k) is a parameterization of U according to k if there are infinitely many problem instances with parameter k.

A large part of the research in parameterized complexity focuses on *fixed parameter tractable* problems, which form the class called FPT. The idea of this class is that the runtime of a parameterized problem (U, k) is polynomial in the length of the input, whereas there may be a factor that is superpolynomial in k. More formally, a parameterized problem (U, k) is in FPT, if, for any k, there is some computable function f such that one can decide (U, k) in time $O(f(k) \cdot n^c)$ for some constant c (that does not depend on k).

A famous example for an NP-hard problem in FPT is the standard parameterization of the vertex cover problem:

DEFINITION 1 (k-VC).

Input: *An unweighted graph G*

Output:

yes *if there is a set of at most k vertices such that each edge is incident to at least one of the vertices in the set,*

no *otherwise.*

If, for a graph G, the answer is "yes", we also say that there is a k-VC in G.

THEOREM 1. *The problem k-VC is in FPT.*

PROOF. To show this theorem, we use an argumentation similar to that in [12]. Observe that for each edge $e = \{u, v\}$ in the input graph G, either u is in the vertex cover or v — there is no other way for e to be incident to the set of vertices forming a solution. But this implies that each solution of size k contains each vertex of degree greater than k: suppose v is a vertex of degree $k + c$ for some constant $c > 0$ and v is not in the solution. Than, according to our observation, all $k + c$ neighboring vertices have to be in the solution. These, however, are too many vertices since we can only choose k vertices in total. If there are more than k vertices of degree higher than k, we know that there is no k-VC in G. After removing the chosen vertices and all its incident edges, the remaining graph has a bounded degree of k. We reduce k by the number of chosen vertices and remove all isolated vertices.

We continue the proof by showing that, if the remaining graph has more than k^2 edges, we are sure that there is no k-VC. Due to the bounded degree, each vertex is incident to at most k edges and therefore k vertices are incident to at most k^2 edges. Hence the remaining graph has at most $2k^2$ vertices. The number of possibilities to choose k vertices from those is $\binom{2k^2}{k}$. Since the runtime to compute a solution in the remaining graph only depends on k, the claimed result immediately follows. $\square$

Note that in the first part of the algorithm runs in polynomial time. Afterwards we either know the result or we are left with a graph of size at most $2k^2$. Such a processing is called *kernelization.* More general, a kernelization is a polynomial time computation that results in a smaller input the size of which only depends on the parameter k but not on the size of the original input.

The concept of kernelization is closely related to the class FPT.

THEOREM 2 (DOWNEY, FELLOWS, STEGE [20]). *A parameterized problem is in FPT if and only if it has a kernelization.* $\square$

We would like to refer to the book "An Invitation to Fixed Parameter Algorithms" from Niedermeier [16] as an excellent introduction into the field of parameterization that focuses on kernelization.

To summarize, the class FPT contains paramterized problems that are not computationally hard according to our notion of hardness, provided that the parameter is a constant. But not all parameterized problems have these nice properties. In fact, there is an infinite hierarchy of so-called W-hardness for parameterized problems. This way, for instance, FPT $= W[0]$, the standard parameterization of the independent set problem (i. e., the problem to decide whether there are k vertices that are pairwise not adjacent) is $W[1]$-hard, and the standard parameterization of the dominating set problem (i. e., the problem to decide whether there are k vertices such that each other vertex in the graph is adjacent to at least one of them) is $W[2]$-hard. To get a rough idea of W-hardness, note that, if FPT $= W[1]$, then any problem in NP can be decided by a deterministic Turing machine in subexponential time. The books from Downey and Fellows [5] and from Flum and Grohe [8] give a detailed introduction into parameterized complexity.

One way to connect parameterization and optimization is to use the concept of efficient polynomial time approximation schemes. An approximation algorithm for a problem U is an algorithm A that computes a solution for U that deviates from an optimal solution by at most a factor α. The value α is the *approximation ratio* of A. More formally, if $\mathrm{Opt}(I)$ is an optimal solution for some input I and $A(I)$ is the solution computed by A, then $A(I)$ is a $c(A(I))/c(\mathrm{Opt}(I))$-approximative solution for I, where c is the cost function of U[1]. The approximation ratio of A is the approximation ratio achieved in the worst case (i. e., minimum approximation ratio over all feasible inputs I for maximization problems and maximum for minimization problems).

An algorithm is a *polynomial time approximation scheme* (PTAS) for U, if for any ε it computes a $1+\varepsilon$ approximation and the runtime is guaranteed to be bounded by a polynomial in the input (say n) and ε^{-1}. Note that, since ε^{-1} is a constant, such a runtime may be very high, since it may have factors such as, for instance, $n^{2^{1/\varepsilon}}$. To avoid such large factors to a certain degree, we consider a restriction of polynomial time approximation schemes that relates to the definition of FPT. If we restrict the runtime to $O(f(\varepsilon^{-1})p(n))$, where f is an arbitrary computable function and p is a polynomial, then an algorithm in this class is an *efficient polynomial time approximation scheme* (EPTAS). This concept was introduced by Cesati and Trevisan [3].

To conclude this section, we give a short introduction into the concept of stability of approximation. The main idea is to split the class of all input instances of a problem into possibly infinitely many subclasses according to their approximability. In other words, we analyze the effect of a small deviation of the specification of feasible inputs on the approximation ratio.

For example let us consider a complete graph $G = (V, E, c)$ with a metric cost function $c : E \to \mathbb{Q}^+$, i. e., $c(\{u, v\}) \leq c(\{u, w\}) + c(\{w, v\})$ for any three pairwise disjoint vertices u, v, and w. We can use Christofides' algorithm (which we will describe more detailed in Section 4) in order to obtain a solution for G that has at most 1.5 times the cost of an optimal solution. But what happens if we relax the restrictions of c? We could, for instance, consider a cost function c_β with relaxed triangle inequality such that $c(\{u, v\}) \leq \beta(c(\{u, w\}) + c(\{w, v\}))$, where $\beta > 1$ is a constant. In this case, Christofides' algorithm does not behave well anymore (for instance, we cannot guarantee that the cost of the computed solution is bounded by a constant factor of the cost of an optimal solution). Using the notion of stability of approximation, we say that Christofides' algorithm is *unstable* according to the distance $\beta - 1$ between the metric function c and the relaxed function c_β. One can, however, modify Christofides' algorithm in order to obtain an algorithm that is *stable* according to the mentioned distance. For details of stability of approximation and an excellent introduction into the study of hard problems we would like to refer to the book "Algorithmics for Hard Problems" of Hromkovič [12].

[1]Note that in some books, the approximation ratio for maximization problems is defined as $c(\mathrm{Opt}(I))/c(A(I))$.

3. HYBRID ALGORITHMS

The previous section focused on identifying structural properties of problems that allow us to guarantee improved solutions according the runtime and the approximation ratio. The instances, however, that do not have these properties, were left behind. In some cases, we can handle problems in such a way that different complexity measures are played against each other such that we can always guarantee some improvements. We could specify, for instance, a single parameter for both exact computations and approximations. According to this parameter we choose either to run an exact algorithm or an approximation algorithm. Our aim is to show that, if the exact algorithm is selected, we can guarantee an improved runtime and if the approximation algorithm is chosen, we can guarantee an improved approximation in polynomial time.

In more general terms, algorithms that select an algorithm from a collection of algorithms based on properties of the input are called *hybrid algorithms.* Using the terminology of hybrid algorithms, the parameter is used by a *selector* S. A selector is basically an algorithm that selects which algorithm of the collection of available algorithms is used for a given input. The origin of this class of algorithms is the so-called *algorithm selection problem* [19].

Here we consider a special type of hybrid algorithms that ensures guarantees for different complexity measures. We can rephrase the description above as follows. Given a hard problem U, we search for a hybrid algorithm A such that the selector S of A chooses a measure m from a set of complexity measures M according to a parameter of the given input. Now we require that, whichever m is chosen by S, the solution computed by A is guaranteed to be "good" according to m. In this context, "good" usually means to be better than what we are able to guarantee in general.

The set M can contain, for example, the worst-case running time, the approximation ratio achievable in polynomial time, or the approximation ratio achievable in linear time.

We continue with illustrating the concept by describing a concrete hybrid algorithm for the problem MaxCut.

Definition 2 (MaxCut).

Input: *An unweighted graph* $G = (V, E)$

Output: *A set of vertices* $V' \subset V$

Cost: *The number of edges between* V' *and* $V \setminus V'$

Goal: *Maximization*

For this problem, we focus on two complexity measures, namely the runtime of an exact algorithm measured in the number of edges and the approximation ratio that we can achieve in linear time. The parameter according to which the selector chooses one of the two measures is the size of a maximal matching in G. Note that a maximal matching is simply a matching such that each of the remaining edge is adjacent to at least one of the edges in the matching. Such a matching can be found in linear time by greedily adding edges. (In contrast, a maximum matching is a largest possible one.)

The currently best exact algorithm for MaxCut measured in the number of edges is from Scott and Sorkin [22] and runs in time $O(2^{m/5})$, where m is the number of edges. Measured in the number of vertices, the best algorithm is from Wiliams [25].

The best approximation algorithm for MaxCut achieves an approximation ratio of 0.87856 [9]. Provided that the unique games conjecture is true, this ratio cannot be improved [13]. The drawback of the mentioned approximation algorithm, however, is that it is based on semidefinite programming. Even though the runtime to solve a semidefinite program is polynomially bounded, it is computationally quite demanding. Therefore it is reasonable to also focus on faster algorithms. The best known algorithm for MaxCut that does not depend on semidefinite programming achieves an approximation ratio of 2 and runs in linear time [21].

In the following theorem, the notation $O^*(n)$ means that polynomial factors are omitted. Apart from that it is identical to the asymptotic upper bound $O(n)$.

Theorem 3 (Vassilevska et al. [24]). *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 $(1/2+\varepsilon/4)$-approximation in linear time.*

Proof. We design a hybrid algorithm A that has the claimed properties. Let $G = (V, E)$ be the graph given as input, where V is the set of vertices and E is the set of edges. Let $|V| = n$ and $|E| = m$. Then the selector of A computes a maximal matching M for G in linear time by repeatedly adding matching edges to M until no edge can be chosen. Now, the selector chooses a complexity measure according to the size of M: if $|M| < \varepsilon m/2$, it chooses to aim for an exact solution and otherwise to aim for an approximation.

We now describe the two algorithms starting with the exact one. The idea is that we only have to find a partition of the vertices in M into two sets that corresponds to an optimal solution. Then we can efficiently deal with the remaining vertices. In order to find an optimal partition of M, we simply try all possibilities in order to keep the one that leads to the largest solution after processing the remaining vertices. Thus, one of the partitions, say M^* and $M \setminus M^*$), corresponds to an optimal solution. Now let us consider the iteration in which M^* is chosen. Now the algorithm has to complete M^* to an optimal solution. To this end, it takes one vertex after the other from $V \setminus M$ and sorts it in in such a way that the number of new edges in the cut is maximized. Let V' be the solution obtained. It remains to show that V' is optimal. To this end note that the vertices in $V \setminus M$ form an independent set since otherwise M is not maximal. Therefore, the vertices from $V \setminus M$ do not interfere with each other: the number of new edges introduced by sorting in one vertex does not depend on where or when the other vertices of $V \setminus M$ are sorted in. In particular, there cannot be a distribution of the vertices from $V \setminus M$ that introduces more edges into the cut then the one chosen greedily.

It is not hard to see that the runtime of the algorithm is dominated by trying all possibilities to partition M into two sets. This number corresponds to the number of subsets of M which is $2^{|M|}$. This leads to a runtime of at most $O^*(2^{\varepsilon m})$ since greedily sorting in the remaining vertices only contributes a polynomial factor.

We finish the proof by describing and analyzing an approximation algorithm that is used if $|M| \geq \varepsilon m/2$.

The main idea of the algorithm is to randomly distribute the vertices in such a way that all edges in M are in the cut. To this end, we first handle the vertices of M separately.

The algorithm repeatedly takes edges $\{u, v\}$ from M and chooses uniformly at random whether u or v is put into V'. Note that this way exactly one vertex of every edge in M is in V'. This ensures that, as mentioned above, each edge from M is in the cut.

The algorithm handles each of the remaining vertices independently. It simply takes one vertex after the other and chooses uniformly at random whether is is in V' or in $V \setminus V'$.

Now we show that the expected number of edges from $E \setminus M$ that are in the cut is $|E \setminus M|/2$. For an edge $e \notin M$, let X_e be the random variable that takes the value 1 if e is in the cut and 0 otherwise. Then, for each $e = \{u, v\}$, the probability that $X_e = 1$ holds is exactly $1/2$: each of the vertices is in V' with a probability of $1/2$. Thus the probability that both are in the same set is $1/4 + 1/4 = 1/2$. Due to linearity of expectation, the expected number of edges in the cut is

$$\sum_{e \in E \setminus M} E[X_e] = \sum_{e \in E \setminus M} 1/2.$$

The total number of edges in the cut is therefore $|M| + |E \setminus M|/2 = \varepsilon m/2 + (m - \varepsilon m/2)/2 = m/2 + \varepsilon m/4$.

The claimed expected approximation ratio now follows immediately, since the cost of an optimal solution is bounded from above by m:

$$\frac{m/2 + \varepsilon m/4}{m} = 1/2 + \varepsilon/4.$$

□

By choosing $\varepsilon < 1/5$, we obtain the claimed properties of the hybrid algorithm: if it computes an optimal solution, the runtime is guaranteed to be improved and if it computes an approximate solution in linear time, the guaranteed approximation ratio is improved.

4. WIN/WIN ALGORITHMS

We now consider the relation between different problems using so-called win/win algorithms. More precisely, given two problems U_1 and U_2 such that both of them have the same set of valid input instances (e. g., both problems take a graph as input), we aim for algorithms that, on a given input, either compute an improved solution for U_1 or for U_2.

Early algorithms of this type appeared in the context of parameterization, mainly used in order to find a kernelization; see [7] for an overview. The name win/win originates from this context and incorporates the idea that one always wins (i. e., one always gets a good kernelization), no matter which input instance is given. This concept naturally translates to approximation and was used in this context in [1, 6, 15].

Note that the concept of win/win algorithms generalizes the type of hybrid algorithms that we considered in Section 3, since we can also see each pair of a problem and a complexity measure as a separate problem.

We now give a few examples of win/win algorithms. To this end, we start with parameterized problems. In [18], Prieto and Sloper presented a kernelization for the k-internal spanning tree problem:

Definition 3 (k-IST).

Input: *An undirected unweighted graph G.*

Output:

- **yes** *if G has a spanning tree with at least k internal vertices,*
- **no** *Otherwise.*

The kernelization of the k-internal-spanning-tree problem uses as its main ingredient a win/win algorithm for k-vertex-cover and k-internal-spanning-tree.

We now show the following win/win result.

Theorem 4 (Prieto, Sloper [18]). *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.*

A main idea of this result is to find a spanning tree in G where the leaves form an independent set. However, we cannot guarantee that we always find such a tree. Instead, we show the following lemma.

Lemma 1 (Prieto, Sloper [18]). *Given a graph $G = (V, E)$, there is a polynomial time algorithm that finds a spanning tree T in G such that either T is a Hamiltonian path or the leaves of T form an independent set in G.*

Proof. We first compute a spanning tree T_1 in G without considering the aimed-for properties. Then we transform T_1 successively in such a way that each time, the number of internal vertices is increased until we get one of the two desired results.

For any $i \geq 1$ such that T_i does not fulfill the claimed conditions, we transform T_i into T_{i+1} as follows. Since the leaves of T_i do not form an independent set, there are two leaves u and v in T_i that are connected by an edge (in E). Let P_i be the unique path in T_i that connects u and v. Since T_i is not a Hamiltonian path, P_i contains vertices that have a degree higher than two. Let $\{s, t\}$ be the edge leading to the first such vertex when starting from u ,i. e., t is the first vertex of a degree higher than 2 and s is its predecessor. Then T_{i+1} is T_i including the edge $\{u, v\}$ but without the edge $\{s, t\}$. Note that T_i is still a tree and still connected, but the structure changed. The vertices u and v are now internal vertices and s is a leaf (unless $u = s$ in which case u does not become an internal vertex).

Since the number of internal vertices is increased in each iteration and cannot be higher than $n - 2$, there are at most $n - 3$ transformation steps, which shows that the algorithm performing the transformation terminates. The outcome is a spanning tree T with the aimed-for properties, because otherwise the algorithm can apply an additional transformation step. (Note that $n - 2$ internal vertices is equivalent with T being a Hamiltonian path.) □

With the preparation done in Lemma 1, Theorem 4 now follows quite naturally.

Proof. (of Theorem 4) Let T be the spanning tree from Lemma 1 and let n be the number of vertices in G. If we obtain a Hamiltonian path, in particular we have found a spanning tree with $n-2$ internal vertices. Since there cannot be a spanning tree with more internal vertices, the result for k-IST is *yes* for $k \leq n - 2$ and *no* for $k = n - 1$ or $k = n$. If the answer is *no*, however, the answer for k-VC is *yes*, because the $n - 2$ internal vertices of T also form a vertex cover.

Otherwise, the leaves of T form an independent set. Note that the internal vertices of T form a vertex cover (all internal edges and the edges between internal vertices and leaves are incident to the internal vertices and there are no edges that connect two leaves). Thus, if there are at most k internal vertices, the answer for k-VC is *yes* and if there are at least k internal vertices, then the answer for k-IST is *yes.* □

We now shift our focus to the use of win/win algorithms in approximation. This type of algorithm was also called *paired approximation* [6], because we start with a pair of problems, obtain a pair of approximative solutions and have the guaranty that one of the solutions is an improved approximation for the corresponding problem. We can see the concept of win/win algorithms for approximation also as a special type of parameterization: the approximation ratio achieved for one of the problems is a parameter for the other one[2].

[2]Note that this type of parameterization does not com-

We illustrate this concept by relating the traveling salesman problem in graphs that have only edges of cost one and two to the independent set problem. The traveling salesman problem is defined as follows:

Definition 4 (TSP).

Input: An edge-weighted complete graph $G = (V, E)$ with cost function c

Output: A cycle C in G that visits each vertex exactly once

Cost: The cost of the cycle, i. e., $c(C)$

Goal: Minimization

If c is an arbitrary cost function, the TSP is hard to approximate. The problem changes considerably, if we restrict the cost function. If edges can only have the cost one or two, we obtain the $(1, 2)$-TSP for which there is a 7/6-approximation algorithm by Papadimitriou and Yannakakis [17]. For the independent set problem, one cannot even design an approximation algorithm with an approximation ratio of $1/n^{1-\varepsilon}$ for any constant ε unless NP=ZPP [10]. The relation of the two problems has an additional motivation: it was used to find a polynomial time approximation scheme (PTAS) for a graph embedding problem in [2].

Theorem 5 (Eppstein [6]). *Let G be a complete graph with edge weights 1 or 2 only and let G' be G without the edges of cost 2. There is a win/win approximation algorithm that, for any $\varepsilon > 0$, either computes a $(1+\varepsilon)$-approximative $(1, 2)$-TSP tour in G or an (ε)-approximative independent set in G'.*

Proof. We first construct a win/win algorithm A that has the claimed properties. Suppose that G' has k components. First, A computes a spanning tree T_i for each component C_i in G' such that either the leaves of T_i form an independent set or T_i is a Hamiltonian path in C_i. We can do this by using Lemma 1. Let I be the set of all leaves the of all computed trees T_i where the leaves form an independent set and one vertex of each remaining component (where T_i is a Hamiltonian path). Note that I is an independent set in G'.

Now A computes a Hamiltonian path P_i for each component C_i in G (i. e., we use some edges of cost 2). To this end, for each T_i that is not a Hamiltonian path, A runs a depth first search starting from a leaf. The path P_i is formed by connecting the vertices in order of the depth first search. This way, there is at most one edge of cost 2 for each leaf except the root and the leaf visited last in the depth first search. (Note that P_i always follows a path within T_i until it reaches a leaf and then it jumps to another vertex in T_i. The jumps may be of cost 1 or 2, but all other edges are of cost 1.) Finally A forms a Hamiltonian tour L in G by including each path P_i and, for each i, inserting an edge of cost 2 between the last vertex of P_i and the first one of P_{i+1} as well as an edge between the last vertex of P_k and the first one of P_1. The output of A is the pair (I, L).

To analyze A, we first compute the cost of L. Let c be the cost function in G. Since G has n vertices, $c(L)$ is n plus the number of edges of cost 2. But these are at most I: each component contributes at least one vertex to I which pays for the connections between the components. Each further edge of cost 2 within some path P_i starts from a vertex from I that was not considered before.

We now analyze the achieved approximation ratios. To this end, let L^* be an optimal Hamiltonian tour in G. Since $c(L^*) \geq n$, if $c(L) \leq (1+\varepsilon)n$, the approximation ratio is bounded by $(1+\varepsilon)n/n = 1+\varepsilon$ and we are done. Otherwise, since $|I| \geq c(L) - n$, $|I| \geq \varepsilon n$. Since no independent set can be larger than n, this directly implies the approximation ratio $\varepsilon n/n = \varepsilon$, which finishes the proof. □

We will now sketch a second pair of approximation ratios. In contrast to the examples before, now we show a relation of two quite similar minimization problems, namely the metric traveling salesman problem (ΔTSP) and the minimum Hamiltonian path problem in metric graphs with prespecified end-vertices (ΔHPP_2). This shows that the relations established due to win/win algorithms are quite different from dual relationships as they appear, for instance, in linear programming.

Both problems run on the same input, namely a metric undirected graph $G = (V, E)$. In this context, a metric graph is a edge-weighted graph where the weight function c obeys the triangle inequality (see Section 2).

Currently, the algorithm of Christofides [4] has the best proven approximation ratio for the ΔTSP, which is 1.5.

The idea of Christofides' algorithm is to compute a minimum spanning tree T of G and a minimum cost perfect matching M on the odd vertices of T. (Remember that in any graph, the number of odd vertices is even.) This way, the multigraph formed by combining T and M has no odd vertices: each odd vertex of T has one additional incident edge. In other words, the graph formed by T and M is Eulerian (since it is even and connected) and thus we can compute an Eulerian tour. Since G is metric, we can shorten the Eulerian tour to a Hamiltonian tour that is not more expensive.

This strategy leads to an approximation ratio of 1.5 by bounding the cost of T and M according to the cost of an optimal Hamiltonian cycle Opt_C in G, $c(\mathrm{Opt}_C)$. To this end, note that any Hamiltonian cycle contains a spanning tree (just remove one arbitrary edge). Therefore we have $c(T) \leq c(\mathrm{Opt}_C)$. To see that $c(M) \leq c(\mathrm{Opt}_C)/2$, suppose that we know Opt_C. Now we can deduct two disjoint perfect matchings from Opt_C as depicted in Figure 1. Due to the triangle inequality, both matichings together do not cost more than $c(\mathrm{Opt}_C)$. This directly implies that at least one of them has not more than half the cost of Opt_C. A minimum cost perfect matching cannot cost more than the matching that we identified. Thus, T and M together cannot cost more than $1.5c(\mathrm{Opt}_C)$, which shows that Christofides' algo-

pletely fit into the usual framework of parameterization, since determining the value of the parameter itself can be a hard problem.

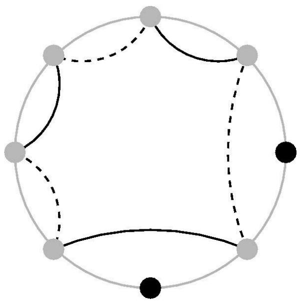

Figure 1: Two disjoint matchings within an optimal Hamiltonian tour. The gray vertices are those that have an odd degree in the spanning tree T.

rithm is 1.5-approximative.

A slight modification of that algorithm was shown by Hoogeveen [11] to be 5/3-approximative for the ΔHPP_2.

As in Christofides' algorithm, we compute a minimum spanning tree T and a matching M. This time, however, we compute a matching on the odd vertices of $T + \{s, t\}$, i. e., we change the parity of the degrees of s and of t. The purpose of this is to obtain a connected graph were all vertices but s and t are even. In such a graph we can efficiently compute an Eulerian path from s to t and — similar to Christofides' algorithm — shorten that path to a Hamiltonian path. As before, T cannot be more expensive than $c(\text{Opt}_P)$, the cost of an optimal Hamiltonian path Opt_P. The challenge is to bound the cost of M, since an optimal path might cost less than an optimal cycle and thus we cannot use the trick with the two disjoint matchings. Instead, we find three disjoint matchings of the odd vertices in a multigraph formed by T and Opt_P, which has a cost of at most $2c(\text{Opt}_P)$. Note that we do not know this graph; we only know that it exists. The path Opt_P has a similar effect as the edge $\{s, t\}$ above, i. e., the parity of the degrees of s and of t is changed and all other parities stay the same. Let C be the even connected graph formed by Opt_P and the unique path connecting s and t in T. Then C shortened to a Hamiltonian cycle contains two disjoint matchings. The third one is contained in the forest that is formed by removing C from $T + \text{Opt}_P$.

The two problems ΔTSP and ΔHPP_2 are strongly related in the sense that, for any given metric graph, either we are guaranteed to obtain an approximation ratio for ΔTSP that is significantly better than 1.5-approximative or we can solve the ΔHPP_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 ΔTSP and the ΔHPP_2 given an input G, u, v and let H_C and H_P be the solutions computed by A. Then we define $\alpha := c(H_P)/c(\text{Opt}_P)$ to be the approximation ratio of the computed Hamiltonian path and $\beta := c(H_C)/c(\text{Opt}_C)$ to be the approximation ratio of the computed Hamiltonian tour.

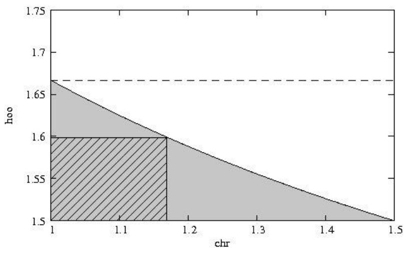

Figure 2: Upper bound on the approximation ratio from Theorem 6. The horizontal line displays the approximation ratio $\alpha \le 5/3$ proven in [11]

THEOREM 6 (MÖMKE [14, 15]). *For any input of A,*

$$\beta \le \min\left\{1.5, \frac{1}{\alpha - 1} - \frac{1}{2}\right\} \quad \textit{and}$$
$$\alpha \le \min\left\{\frac{5}{3}, \frac{1}{\beta + 1/2} + 1\right\}.$$

□

The main idea of the proof is find two disjoint matchings along an optimal Hamiltonian path in G. One of these matchings is for the cycle, the other one for the path.

Note that $\alpha = 5/3$ implies $\beta = 1$. Thus, each worst-case instance of Hoogeveen's algorithm allows us to compute an optimal Hamiltonian tour.

5. CONCLUSION

We have seen several approaches to use structural properties to facilitate the computation. The concept of win/win algorithms extends the use of such properties in a nice way, since the unability to solve a problem becomes a useful property that we can use to solve a different problem.

We have seen some examples of pairs of problems that allow to apply the concept of win/win algorithms successfully. One may now ask whether this is always the case. But we have to answer this question negatively, since Eppstein showed in [6] that as well for the pair of the independent set problem and the maximum clique problem as for the pair of the set cover problem and the hitting set problem, there are no win/win algorithms that lead to improved approximations unless P = NP.

Since this field of of win/win algorithms for approximation is very new, there is a huge number of pairs of problems one may consider. In other words, there is a whole menu of open problems involved in analyzing which pairs of problems allow win/win algorithms and which ones can be shown to not allow win/win algorithms.

Acknowledgement

This survey was partially funded by ERC Advanced investigator grant 226203.

6. REFERENCES

[1] H.-J. Böckenhauer, R. Klasing, T. Mömke, and M. Steinová. Improved approximations for TSP with simple precedence constraints. In T. Calamoneri and J. Díaz, editors, *Proc. of the 7th International Conference on Algorithms and Complexity (CIAC 2010)*, volume 6078 of *Lecture Notes in Computer Science*, pages 61–72. Springer-Verlag, 2010.

[2] S. Cabello, D. Eppstein, and S. Klavžar. The Fibonacci dimension of a graph. *CoRR*, abs/0903.2507, 2009.

[3] M. Cesati and L. Trevisan. On the efficiency of polynomial time approximation schemes. *Information Processing Letters*, 64(4):165–171, 1997.

[4] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University, 1976.

[5] R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Monographs in Computer Science. Springer-Verlag, New York, 1999.

[6] D. Eppstein. Paired approximation problems and incompatible inapproximabilities. In M. Charikar, editor, *Proc. of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2010)*, pages 1076–1086. Society for Industrial and Applied Mathematics, 2010.

[7] M. R. Fellows. Blow-ups, win/win's, and crown rules: Some new directions in FPT. In H. L. Bodlaender, editor, *Proc. of the 29th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2003)*, volume 2880 of *Lecture Notes in Computer Science*, pages 1–12, Berlin, 2003. Springer-Verlag.

[8] J. Flum and M. Grohe. *Parameterized Complexity Theory.* Springer-Verlag, 2006.

[9] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6):1115–1145, 1995.

[10] J. Håstad. Clique is hard to approximate within $n^{1\text{-epsilon}}$. *Electronic Colloquium on Computational Complexity (ECCC)*, 4(38), 1997.

[11] J. A. Hoogeveen. Analysis of Christofides' heuristic: some paths are more difficult than cycles. *Operations Research Letters*, 10(5):291–295, 1991.

[12] J. Hromkovič. *Algorithmics for Hard Problems. Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics.* Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 2003.

[13] S. Khot, G. Kindler, E. Mossel, and R. O'Donnell. Optimal inapproximability results for max-cut and other 2-variable CSPs? In *Proc. of the 45th Annual Symposium on Foundations of Computer Science (FOCS 2004)*, pages 146–154. IEEE, 2004.

[14] T. Mömke. Structural properties of hard metric TSP inputs. Technical Report 685, ETH Zürich, 2010.

[15] T. Mömke. Structural properties of hard metric TSP inputs. In *Proc. of the 37th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2011)*, Lecture Notes in Computer Science, Berlin, 2011. Springer-Verlag. To appear.

[16] R. Niedermeier. *Invitation to Fixed Parameter Algorithms (Oxford Lecture Series in Mathematics and Its Applications).* Oxford University Press, USA, March 2006.

[17] C. H. Papadimitriou and M. Yannakakis. The traveling salesman problem with distances one and two. *Mathematics of Operations Research*, 18(1):1–11, 1993.

[18] E. Prieto and C. Sloper. Either/or: using vertex cover structure in designing FPT-algorithms—the case of k-Internal Spanning Tree. In F. K. H. A. Dehne, J.-R. Sack, and M. H. M. Smid, editors, *Proc. of the 8th International Workshop on Algorithms and Data Structures (WADS 2003)*, volume 2748 of *Lecture Notes in Computer Science*, pages 474–483, Berlin, 2003. Springer-Verlag.

[19] J. R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976.

[20] U. S. Rodney G. Downey, Michael R. Fellows. Parameterized complexity: A framework for systematically confronting computational intractability. In *Contemporary Trends in Discrete Mathematics: From DIMACS and DIMATIA to the Future*, pages 49–99, 1999.

[21] S. Sahni and T. F. Gonzalez. P-complete approximation problems. *Journal of the ACM*, 23(3):555–565, 1976.

[22] A. D. Scott and G. B. Sorkin. Faster algorithms for MAX CUT and MAX CSP, with polynomial expected time for sparse instances. In S. Arora, K. Jansen, J. D. P. Rolim, and A. Sahai, editors, *Proc. of the 7th International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM 2003)*, volume 2764 of *Lecture Notes in Computer Science*, pages 382–395. Springer-Verlag, 2003.

[23] D. A. Spielman and S.-H. Teng. Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time. In *Proc. of the 33th Annual ACM Symposium on Theory of Computing (STOC 2001)*, pages 296–305. ACM Press, 2001.

[24] V. Vassilevska, R. Williams, and S. L. M. Woo. Confronting hardness using a hybrid approach. In *Proc. of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2006)*, pages 1–10, New York, 2006. Society for Industrial and Applied Mathematics.

[25] R. Williams. A new algorithm for optimal constraint satisfaction and its implications. In J. Díaz, J. Karhumäki, A. Lepistö, and D. Sannella, editors, *Proc. of the 31st International Colloquium on Automata, Languages and Programming (ICALP 2004)*, volume 3142 of *Lecture Notes in Computer Science*, pages 1227–1237. Springer-Verlag, 2004.