Stability of Approximation Algorithms for Hard Optimization Problems

Juraj Hromkovič

Dept. of Computer Science I (Algorithms and Complexity), RWTH Aachen
Ahornstraße 55, 52056 Aachen, Germany
jh@I1.informatik.rwth-aachen.de

Abstract. To specify the set of tractable (practically solvable) computing problems is one of the few main research tasks of theoretical computer science. Because of this the investigation of the possibility or the impossibility to efficiently compute approximations of hard optimization problems becomes one of the central and most fruitful areas of recent algorithm and complexity theory. The current point of view is that optimization problems are considered to be tractable if there exist polynomial-time randomized approximation algorithms that solve them with a reasonable approximation ratio. If a optimization problem does not admit such a polynomial-time algorithm, then the problem is considered to be not tractable.

The main goal of this paper is to relativize this specification of tractability. The main reason for this attempt is that we consider the requirement for the tractability to be strong because of the definition of the complexity as the "worst-case" complexity. This definition is also related to the approximation ratio of approximation algorithms and then an optimization problem is considered to be intractable because some subset of problem instances is hard. But in the practice we often have the situation that the hard problem instances do not occur. The general idea of this paper is to try to partition the set of all problem instances of a hard optimization problem into a (possibly infinite) spectrum of subclasses according to their polynomial-time approximability. Searching for a method enabling such a fine problem analysis (classification of problem instances) we introduce the concept of stability of approximation. To show that the application of this concept may lead to a "fine" characterization of the hardness of particular problem instances we consider the traveling salesperson problem and the knapsack problem.

1 Introduction

Historically, the first informally formulated questions about computability and decidability dates back at the turn of the century, when the great mathematican David Hilbert asked for the possibility to encode all mathematical problems in some suitable formalism and to determine (decide) their truth by an algorithm. The landmark paper by Kurt Gödel [11] showed that this is impossible. Moreover, this paper led to the formal definition of the notion of algorithm and so put the fundamentals of theoretical computer science. So, it was possible to formaly pose questions of the following kind:

"Is a given problem algorithmically solvable (computable, decidable)?"

This led to the development of the computability theory (for a nice overview see [22]), where mathematical methods determining the existence or non-existence of algorithms for different mathematical problems were created. With the development of computer technologies a new, more involved question than the original one becomes the central one:

"If a problem is computable, how much physical (computer) work is necessary and sufficient to algorithmically solve the problem?"

This question initialized the origin of the algorithm and complexity theory in the sixties. Measuring the complexity of algorithms the question "What is computable?" was replaced by the question

"What is practically computable?"

But to answer this question one has first to specify (formalize) the meaning of the notion *"practically computable"*, called **tractable** in the current literature. The history of the development of complexity and algorithm theory can be more or less viewed as the history of the development of our opinion on the specification of tractability.

The first specification was done already in the sixties, when people decided to consider a computing problem to be tractable if there exists an algorithm that solves the problem in polynomial-time in the input length. A problem is called intractable if it does not admit any polynomial-time algorithm. The main reasons for this decision were the following two:

(i) (*a more theoretical reason*) The class of polynomial-time computations is robust in that sense that its specification is independent on the computing model (formalism) chosen, if the model is reasonable for the complexity measurement.[1]

(ii) (*a more practical reason*) If a problem had a polynomial-time algorithm, then usually the people were able to find an at most $O(n^3)$ time algorithm for it. Such an algorithm is really practical because one can use it to solve problem instances of sizes that realistically appear in the practice. On the other hand, if the complexity of an algorithm is 2^n, where n is the input size, then already for realistic input size $n = 100$ 2^n is a 31-digit number. Since the number of microseconds since the "Big Bang" should has 24 digits, it is clear that there is no chance to apply this algorithm.

Considering this classification (specification of tractability) the main problem becomes to find a method for proving that a computing problem does not admit

[1] Note, that this kind of robustness is a reasonable requirement that should be fulfilled for every attempt to define the class of tractable problems.

any polynomial-time algorithm, i.e., that it is intractable. Because no mathematical proof of intractability for a language from NP has been realized up till now, and the most interesting problems in practice have their decision versions in NP, Cook introduced the concept of NP-completeness in his seminal paper [9]. This concept enabled, under the assumpion P $\neq$ NP, to prove intractability of thousands of computing problems.

Immediately after introducing NP-hardness (completeness) [9] as a concept for proving intractability of computing problems [16], the following question has been posed:

"If an optimization problem does not admit any efficiently computable optimal solution, is there a possibility to efficiently compute at least an approximation of the optimal solution?"

Several researchers (see, for instance [15,17,7,14]) provided already in the middle of the seventies a positive answer for some optimization problems. It may seem to be a fascinating effect if one jumps from the exponential complexity (a huge inevitable amount of physical work) to the polynomial complexity (tractable amount of physical work) due to a small change in the requirement – instead of an exact optimal solution one forces a solution whose quality differs from the quality of an optimal solution at most by $\varepsilon \cdot 100\ \%$ for some ε. This effect is very strong, especially, if one considers problems for which this approximation concept works for any small relative difference ε (see the concept of approximation schemes in [14,19,21,4]).

On the other hand we know several computing problems that admit efficient polynomial-time randomized algorithms, but no polynomial-time deterministic algorithm is known for them.[2] Usually the randomized algorithms are practical because the probability to compute the right output can be increased to $1 - \delta$ for any small $\delta > 0$.

These are the reasons why currently optimization problems are considered to be tractable if there exist randomized polynomial-time approximation algorithms that solve them with a reasonable approximation ratio. In what follows an α-approximation algorithm for a minimization [maximization] problem is an algorithm that provides feasible solutions whose cost divided by the cost of optimal solutions is at most α [is at least $\frac{1}{\alpha}$].

There is also another possibility to jump from NP to P. Namely, to consider the subset of inputs with a special, nice property instead of the whole set of inputs for which the problem is well-defined. A nice example is the Traveling Salesperson Problem (TSP). TSP is not only NP-hard, but also the search of an approximation solution for TSP is NP-hard for every ε.[3] But if one considers TSP for inputs satisfying the triangle inequality (so called Δ-TSP), one can even design an $\frac{3}{2}$-approximation algorithm [7]. The situation is still more

[2] Currently, we do not know any randomized bounded-error polynomial-time algorithm for an NP-hard problem.

[3] In fact, under the assumtion P $\neq$ NP, there is no $p(n)$-approximation algorithm for TSP for any polynomial p.

interesting, if one considers the Euclidean TSP, where the distances between the nodes correspond to the distances in the Euclidean metrics. The Euclidean TSP is NP-hard [20], but for every small $\varepsilon > 0$ one can design a polynomial-time $(1+\varepsilon)$-approximation algorithm [2,3,18].[4]

Exactly this second approach for jumping from NP to P is the reason why we propose again to revise the notion of tractability of optimization problems. This approach shows that the main drawback of the current definition of tractability is in the definition of complexity and approximality in the worst-case manner. Because of some hard problem instances a problem is considered to be intractable. But if one can specify the border between the hard problem instances and the easy ones, and one can efficiently decide the membership of a given instance to one of this two classes, then this computing problem may finally be considered to be tractable in several applications. Our general idea is to try to split the set of all input sequences of the given problem into possible infinitely many subclasses according to the hardness of their polynomial-time approximability. To provide a method that is able to achieve this goal, we introduce the concept of approximation stability.

Informally, one can describe the idea of our concept by the following scenario. One has an optimization problem P for two sets of input instances L_1 and L_2, $L_1 \subset L_2$. For L_1 there exists a polynomial-time α-approximation algorithm A, but for L_2 there is no polynomial-time γ-approximation algorithm for any $\gamma > 0$ (if NP is not equal to P). We pose the following question: Is the algorithm A really useful for inputs from L_1 only? Let us consider a distance measure M in L_2 determining the distance $M(x)$ between L_1 and any $x \in L_2$. Now, one can consider an input $x \in L_2 - L_1$, with $M(x) \leq k$ for some positive real k. One can look for how "good" the algorithm A is for the input $x \in L_2 - L_1$. If for every $k > 0$ and every x with the distance at most k to L_1, A computes an $\delta_{\alpha,k}$ approximation of an optimal solution for x ($\delta_{\alpha,k}$ is considered to be a constant depending on k and α only), then one can say that A is "(approximation) stable" according to M.

Obviously, the idea of the concept of approximation stability is similar to that of stability of numerical algorithms. But instead of observing the size of the change of the output value according to a small change of the input value, we look for the size of the change of the approximation ratio according to a small change in the specification (some parameters, characteristics) of the set of input instances considered. If the exchange of the approximation ratio is small for every small change in the specification of the set of input instances, then we have a stable algorithm. If a small change in the specification of the set of input instances causes an essential (including the size of the input instances) increase of the relative error, then the algorithm is unstable.

[4] Obviously, we know a lot if similar examples where with restricting the set of inputs one crosses the border between decidability and undecidability (Post Correspondence Problem) or the border between P and NP (SAT and 2-SAT, or vertex cover problem).

The concept of stability enables to show positive results extending the applicability of known approximation algorithms. As we shall see later, the concept motivates us to modify an unstable algorithm A in order to get a stable algorithm B that achieves the same approximation ratio on the original set of input instances as A has, but B can be successfully used also outside of the original set of input instances. This concept seems to be useful because there are many problems for which an additional assumption on the "parameters" of the input instances leads to an essential decrease of the hardness of the problem. Thus, such "effects" are the starting points for trying to partition the whole set of input instances into a spectrum of classes according to approximability.

This paper is organized as follows: In Section 2 we present our concept of approximation stability. Sections 3 and 4 illustrates the usefulness of this concept by applying it to TSP and the knapsack problem respectively. Section 5 is devoted to a general discussion about possible applications of this concept.

2 The Concept of the Stability of Approximation

We assume that the reader is familiar with the basic concepts and notions of algorithmics and complexity theory as presented in standard textbooks like [4,10,13,21,23]. Next, we give a new (a little bit revised) definition of the notion of an optimization problem. The reason to do this is to obtain the possibility to study the influence of the input sets on the hardness of the problem considered. Let $\mathbb{N} = \{0, 1, 2, \dots\}$ be the set of nonnegative integers, and let $\mathbb{R}^+$ be the set of positive reals.

Definition 1. *An* ***optimization problem*** *U is an 7-tuple $U = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal)$, where*

(i) Σ_I is an alphabet called ***input alphabet****,*
(ii) Σ_O is an alphabet called ***output alphabet****,*
(iii) $L \subseteq \Sigma_I^*$ is a language over Σ_I called the ***language of consistent inputs****,*
(iv) $L_I \subseteq L$ is a language over Σ_I called the ***language of actual inputs****,*
(v) $\mathcal{M}$ is a function from L to $2^{\Sigma_O^*}$, where, for every $x \in L$, $\mathcal{M}(x)$ is called the ***set of feasible solutions*** *for the input x,*
(vi) cost is a function, called ***cost function****, from $\bigcup_{x \in L} \mathcal{M}(x) \times L_I$ to $\mathbb{R}^+$,*
(vii) $goal \in \{minimum,\ maximum\}$.

For every $x \in L$, we define

$$\boldsymbol{Output_U(x)} = \{y \in \mathcal{M}(x) \mid cost(y) = goal\{cost(z) \mid z \in \mathcal{M}(x)\}\},$$

and

$$\boldsymbol{Opt(x)} = cost(y) \text{ for some } y \in Output_U(x).$$

Clearly, the meaning for Σ_I, Σ_O, $\mathcal{M}$, *cost* and *goal* is the usual one. L may be considered as a set of consistent inputs, i.e., the inputs for which the optimization problem is consistently defined. L_I is the set of inputs considered and only these inputs are taken into account when one determines the complexity of the optimization problem U. This kind of definition is useful for considering the complexity of optimization problems parametrized according to their languages of actual inputs. In what follows ***Language*(*U*)** denotes the language L_I of actual inputs of U.

To illustrate Definition 1 consider the Knapsack Problem (KP). In what follows ***bin*(*u*)** denotes the integer binary coded by the string $u \in \{0,1\}^*$. The input of KP consists of $2n+1$ integers $w_1, w_2, \ldots, w_n, b, c_1, c_2, \ldots, c_n, n \in \mathbb{N}$. So, one can consider $\Sigma_I = \{0,1,\#\}$ with the binary coding of integers and $\#$ for ",". The output is a vector $x \in \{0,1\}^n$, and so we set $\Sigma_O = \{0,1\}$. $L = \{0,1\}^* \cdot \bigcup_{i=0}^{\infty} (\#\{0,1\}^*)^{2i}$. We speak about the Simple Knapsack Problem (SKP) if $w_i = c_i$ for every $i = 1, \ldots, n$. So, we can consider $L_I = \{z_1\#z_2\#\ldots\#z_n\#b\#z_1\#z_2\#\ldots\#z_n \mid b,\ z_i \in \{0,1\}^* \text{ for } i = 1, \ldots, n,\ n \in \mathbb{N}\}$ as a subset of L. $\mathcal{M}$ assigns to each $I = y_1\#\ldots\#y_n\#b\#u_1\#\ldots\#u_n$ the set of words $\mathcal{M}(I) = \{x = x_1x_2\ldots x_n \in \{0,1\}^n \mid \sum_{i=1}^{n} x_i \cdot bin(y_i) \leq bin(b)\}$. For every $x = x_1 \ldots x_n \in \mathcal{M}(I)$, $cost(x) = \sum_{i=1}^{n} x_i \cdot bin(u_i)$. The goal is maximum. So, $KP = (\Sigma_I, \Sigma_O, L, L, \mathcal{M}, cost, goal)$, and $SKP = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal)$.

Definition 2. *Let $U = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal)$ be an optimization problem. We say that an algorithm A is a* ***consistent algorithm for U*** *if, for every input $x \in L_I$, A computes an output $A(x) \in \mathcal{M}(x)$. We say that* ***A solves U*** *if, for every $x \in L_I$, A computes an output $A(x)$ from $Output_U(x)$. The time complexity of A ist defined as the function*

$$Time_A(n) = \max\{Time_A(x) \mid x \in L_I \cap \Sigma_I^n\}$$

from $\mathbb{N}$ to $\mathbb{N}$, where $Time_A(x)$ is the length of the computation of A on x.

Next, we give the definitions of standard notions in the area of approximation algorithms (see e.g. [8,13]).

Definition 3. *Let $U = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal)$ be an optimization problem, and let A be a consistent algorithm for U. For every $x \in L_I$, the* ***relative error*** $\boldsymbol{\varepsilon_A(x)}$ *is defined as*

$$\boldsymbol{\varepsilon_A(x)} = \frac{|cost(A(x)) - Opt(x)|}{Opt(x)}.$$

For any $n \in \mathbb{N}$, we define ***the relative error of A***

$$\boldsymbol{\varepsilon_A(n)} = \max\{\varepsilon_A(x) \mid x \in L_I \cap \Sigma_I^n\}.$$

For every $x \in L_I$, the ***approximation ratio*** $\boldsymbol{R_A(x)}$ *is defined as*

$$\boldsymbol{R_A(x)} = \max\left\{\frac{cost(A(x))}{Opt(x)}, \frac{Opt(x)}{cost(A(x))}\right\} = 1 + \varepsilon_A(x).$$

For any $n \in \mathbb{N}$, we define the ***approximation ratio of A*** *as*

$$\boldsymbol{R_A(n)} = \max\{R_A(x) \mid x \in L_I \cap \Sigma_I^n\}.$$

For any positive real $\delta > 1$, we say that A is an ***δ-approximation algorithm for U*** *if $R_A(x) \le \delta$ for every $x \in L_I$.*

For every function $f : \mathbb{N} \to \mathbb{R}^+$, we say that A is a ***$f(n)$-approximation algorithm for U*** *if $R_A(n) \le f(n)$ for every $n \in \mathbb{N}$.*

The best what can happen for a hard optimization problem U is that one has a polynomial-time $(1+\varepsilon)$-approximation algorithm for U for any $\varepsilon > 0$. In that case we call this collection of approximation algorithms a ***polynomial-time approximation scheme*** (**PTAS**) ***for U***. A nicer definition of a PTAS than the above one is the following one. An algorithm B is a PTAS for an optimization problem U if B computes an output $B(x,\varepsilon) \in \mathcal{M}(x)$ for every input $(x,\varepsilon) \in Language_U \times \mathbb{R}^+$ with

$$\frac{|cost(B(x,\varepsilon)) - Opt(x)|}{Opt(x)} \le \varepsilon$$

in time polynomial according to $|x|$. If the time complexity of B can be bounded by a function that is polynomial in both $|x|$ and ε^{-1}, then we say that B is a ***fully polynomial-time approximation schema*** **(FPTAS)** for U.

Now, we define the complexity classes of optimization problems in the usual way (see e.g. [13,19]).

Definition 4.

$$\begin{aligned}\boldsymbol{NPO} = \{\ U = (&\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal) \mid \\ &U \text{ is an optimization problem}, \quad L, L_I \in P; \\ &\text{for every } x \in L, \mathcal{M}(x) \in P; cost \text{ is computable in polynomial time}\},\end{aligned}$$

For every optimization problem $U = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal)$, the ***underlying language of U*** *is*

$$Under_U = \{(w,k) \mid w \in L_I, k \in \mathbb{N} - \{0\}, Opt_U(w) \le k\}$$

if goal = maximum. Analogously, if goal = minimum

$$Under_U = \{(w,r) \mid w \in L_I,\ r \in \mathbb{N} - \{0\},\ Opt_U(w) \ge r\}.$$

$\boldsymbol{PO} = \{U \in NPO \mid Under_U \in P\}$
$\boldsymbol{APX} = \{U \in NPO \mid$ *there exists an ε-approximation algorithm for U for some $\varepsilon \in \mathbb{R}^+$*$\}$.

In order to define the notion of stability of approximation algorithms we need to consider something like a distance between a language L and a word outside L.

Definition 5. *Let* $U = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal)$ *and* $\overline{U} = (\Sigma_I, \Sigma_O, L, L, \mathcal{M}, cost, goal)$ *be two optimization problems with* $L_I \subset L$. *A* ***distance function for U according to*** $\boldsymbol{L_I}$ *is any function* $h : L \to \mathbb{R}^+$ *satisfying the properties*

(i)
$$h(x) = 0 \textit{ for every } x \in L_I.$$

(ii)
$$h \textit{ can be computed in polynomial time.}$$

We define, for any $r \in \mathbb{R}^+$,

$$\boldsymbol{Ball_{r,h}(L_I)} = \{w \in L \mid h(w) \le r\}.$$

Let A *be a consistent algorithm for* $\overline{U}$, *and let* A *be an* ε*-approximation algorithm for* U *for some* $\varepsilon \in \mathbb{R}^+$. *Let* p *be a positive real. We say that* A *is* ***p-stable according to h*** *if, for every real* $0 \le r \le p$, *there exists a* $\delta_{r,\varepsilon} \in \mathbb{R}^+$ *such that* A *is an* $\delta_{r,\varepsilon}$*-approximation algorithm for* $U_r = (\Sigma_I, \Sigma_O, L, Ball_{r,h}(L_I), \mathcal{M}, cost, goal)$.[5]

A is ***stable according to h*** *if A is p-stable according to h for every* $p \in \mathbb{R}^+$. *We say that A is* ***unstable according to h*** *if A is not p-stable for any* $p \in \mathbb{R}^+$.

For every positive integer r, *and every function* $f_r : \mathbb{N} \to \mathbb{R}^+$ *we say that* A *is* $\boldsymbol{(r, f(n))}$***-quasistable according to h*** *if* A *is an* $f_r(n)$*-approximation algorithm for* $U_r = (\Sigma_I, \Sigma_O, L, Ball_{r,h}(L_I), \mathcal{M}, cost, goal)$.

We see that the existence of a stable c-approximation algorithm for U immediately implies the existence of a $\delta_{r,c}$-approximation algorithm for U_r for any $r > 0$. Note, that applying the stability to PTASs one can get two different outcomes. Consider a PTAS A as a collection of polynomial-time $(1+\varepsilon)$-approximation algorithms A_ε for every $\varepsilon \in \mathbb{R}^+$. If A_ε is stable according to a distance measure h for every $\varepsilon > 0$, then we can obtain either

(i) a PTAS for $U_r = (\Sigma_I, \Sigma_O, L, Ball_{r,h}(L_I), \mathcal{M}, cost, goal)$ for every $r \in \mathbb{R}^+$ (this happens, for instance, if $\delta_{r,\varepsilon} = 1 + \varepsilon \cdot f(r)$, where f is an arbitrary function), or
(ii) a $\delta_{r,\varepsilon}$-approximation algorithm for U_r for every $r \in \mathbb{R}^+$, but no PTAS for U_r for any $r \in \mathbb{R}^+$ (this happens, for instance, if $\delta_{r,\varepsilon} = 1 + r + \varepsilon$).

To capture these two different situations we introduce the notion of "superstability" as follows:

Definition 6. *Let* $U = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal)$ *and* $\overline{U} = (\Sigma_I, \Sigma_O, L, L, \mathcal{M}, cost, goal)$ *be two optimization problems with* $L_I \subset L$. *Let* h *be a distance function for* $\bar{U}$ *according to* L_I, *and let* $U_r = (\Sigma_I, \Sigma_O, L, Ball_{r,h}(L_I), \mathcal{M}, cost, goal)$ *for every* $r \in \mathbb{R}^+$. *Let* $A = \{A_\varepsilon\}_{\varepsilon > 0}$ *be a PTAS for* U.

If, for every $r > 0$ *and every* $\varepsilon > 0$, A_ε *is a* $\delta_{r,\varepsilon}$*-approximation algorithm for* U_r, *we say that the PTAS A is* **stable according to *h***.

[5] Note, that $\delta_{r,\varepsilon}$ is a constant depending on r and ε only.

If $\delta_{r,\varepsilon} \leq f(\varepsilon) \cdot g(r)$*, where*

(i) f *and* g *are some functions from* $\mathbb{R}^{\geq 0}$ *to* $\mathbb{R}^{+}$*, and*
(ii) $\lim_{\varepsilon \to 0} f(\varepsilon) = 0$*,*

then we say that the PTAS A *is* **super-stable according to *h***.

Remark 1. If A is a super-stable (according to a distance function h) PTAS for an optimization problem $U = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal)$, then A is a PTAS for the optimization problem $U_r = (\Sigma_I, \Sigma_O, L, Ball_{r,h}(L_I), \mathcal{M}, cost, goal)$ for any $r \in \mathbb{R}^{+}$.

One may see that the notions of stability can be useful for answering the question how broadly a given approximation algorithm is applicable. So, if one is interested in positive results then one is looking for a suitable distance measure that enables to use the algorithm outside the originally considered set of inputs. In this way one can search for the border of the applicability of the given algorithm. If one is interested in negative results then one can try to show that for any reasonable distance measure the considered algorithm cannot be extended to work for a much larger set of inputs than the original one. In this way one can search for fine boundaries between polynomial approximability and polynomial non-approximability.

3 Stability of Approximation and Traveling Salesperson Problem

We consider the well-known TSP problem that is in its general form very hard for approximation. But if one considers complete graphs in which the triangle inequality holds, then we have the two following approximation algorithms for the Δ-TSP.

Algorithm ST (Spanning Tree)

Input: a complete graph $G = (V, E)$, and a cost function $c : E \to \mathbb{N}^{+}$ satisfying the triangle inequality ($c(\{u, v\}) \leq c(\{v, w\}) + c(\{w, u\})$ for all three pairwise different $u, v, w \in V$).
Step 1: Construct a minimal spanning tree T of G according to c.
Step 2: Choose an arbitrary vertex $v \in V$. Realize Depth-first-search of T from v, and order the vertices in order in that they are visited. Let H be the resulting sequence.
Output: The Hamiltonian tour $\bar{H} = H, v$.

Christofides Algorithm

Input: a complete graph $G = (V, E)$, and a cost function $c : E \to \mathbb{N}^+$ satisfying the triangle inequality.
Step 1: Construct a minimal spanning tree T of G according to c.
Step 2: $S := \{v \in V \mid deg_T(v) \text{ is odd}\}$.
Step 3: Compute a minimum-weight perfect matching M on S in G.
Step 4: Create the multigraph $G' = (V, E(T) \cup M)$ and construct an Euler tour w in G'.
Step 5: Construct a Hamiltonian tour H of G by shortening w {realize it by removing all repetitions of the occurences of every vertex in w, in one run via w from the left to the right}.
Output: H

It is well known that the algorithm ST and the Christofides algorithm are α-approximation algorithms for the Δ-TSP with $\alpha = 2$ and $\alpha = \frac{3}{2}$ respectively. It can be simply observed that both algorithms are consistent for the general TSP. Since the triangle inequality is crucial for the approximability of the problem instances of Δ-TSP, we consider the following distance functions.

We define for every $x \in L$,

$$dist(x) = \max\left\{0, \max\left\{\frac{c(\{u,v\})}{c(\{u,p\}) + c(\{p,v\})} - 1 \,\middle|\, u,v,p \in V_x\right\}\right\},$$

and

$$distance(x) = \max\left\{0, \max\left\{\frac{c(\{u,v\})}{\sum_{i=1}^{m} c(\{p_i, p_{i+1}\})} - 1 \,\middle|\, u,v \in V_x, \right.\right.$$

$$\left.\left.\text{and } u = p_1, p_2, \ldots, p_{m+1} = v \text{ is a simple path between } u \text{ and } v \text{ in } G_x\right\}\right\}$$

We observe that *dist* and *distance* measure the "degree" of violating the triangle inequality in two different ways. Let $x = (G, c)$ be an input instance of TSP. For the simplicity we consider the size of x as the number of nodes of G instead of the real length of the code of x over Σ_I. The inequality $dist(G, c) \le r$ implies

$$c(\{u,v\}) \le (1 + r) \cdot [c(\{u,w\}) + c(\{w,v\})]$$

for all pairwise different vertices u, v, w of G. The measure *distance* involves a much harder requirement than *dist*. If $distance(G, c) \le r$, then $c(\{u,v\})$ may not be larger than $(1 + r)$ times the cost of any path between u and v[6]. The next results show that these two distance functions are really different because the approximation algorithms for Δ-TSP considered above are stable according to *distance* but not according to *dist*.

Lemma 1. *The algorithm SP, and the Christofides algorithm are stable according to distance.*

[6] than the cost of the shortest path between u and v

Proof. We present the proof for the algorithm SP only. Let $x \in Ball_{r,distance}(L_I)$ for an $r \in \mathbb{R}^+$. Let D_x be the Eulerian tour corresponding to the moves of DFS in Step 2. Observe that D_x goes twice via every edge of the minimal spanning tree T. Since the cost of T is smaller than the cost of any optimal Hamiltonian tour, the cost of D_x is at most twice the cost of an optimal Hamiltonian tour. Let $H_x = v_1, v_2, \ldots, v_n, v_1$ be the resulting Hamiltonian tour. Obviously, D_x can be written as $v_1 P_1 v_2 P_2 v_3 \ldots v_n P_n v_1$, where P_i is a path between v_i and $v_{(i+1) \bmod n}$ in D_x. Since $c(\{v_i, v_{(i+1) \bmod n}\})$ is at most $(1+r)$ times the cost of the path $v_i P_i v_{(i+1) \bmod n}$ for all $i \in \{1, 2, \ldots, n\}$, the cost for H_x is at most $(1+r)$ times the cost for D_x. Since the cost of D_x ist at most $2 \cdot Opt(x)$, the algorithm SP is a $(2 \cdot (1+r))$-approximation algorithm for $(\Sigma_I, \Sigma_O, L, Ball_{r,distance}(L_I), \mathcal{M}, cost, minimum)$. □

The next result shows that our approximation algorithms provide a very week approximation for input instances of Δ_{1+r}-TSP, where Δ_{1+r}-TSP $= (\Sigma_I, \Sigma_O, L, Ball_{r,dist}(L_\Delta), \mathcal{M}, cost, minimum)$.

Lemma 2. *For every $r \in \mathbb{R}^+$, the Christofides algorithm is $\left(r, \frac{3}{2}(1+r)^{\lceil \log_2 n \rceil}\right)$-quasistable for dist, and the algorithm ST is $\left(r, 2 \cdot (1+r)^{\lceil \log_2 n \rceil}\right)$-quasistable for dist.*

Proof. Again, we realize the proof for the algorithm ST only. Let $x = (G, c) \in Ball_{r,dist}(L_\Delta)$ for an $r \in \mathbb{R}^+$. Let T, D_x, and H_x have the same meaning as in the proof of Lemma 1. The crucial idea is the following one. To exchange a path v, P, u of a length m, $m \in \mathbb{N}^+$, for the edge $\{v, u\}$ one can proceed as follows. For any $p, s, t \in V(G)$ one can exchange the path p, s, t for the edge $\{p, t\}$ by the cost increase bounded by the multiplicative constant $(1+r)$. This means that reducing the length m of a path to the length $\lceil m/2 \rceil$ increases the cost of the connection between u and v by at most $(1+r)$-times. After at most $\lceil \log_2 m \rceil$ such reduction steps one reduces the path v, P, u to the path v, u and

$$cost(u, v) = c(\{v, u\}) \le (1+r)^{\lceil \log_2 m \rceil} \cdot cost(v, P, u) \ .$$

Following the argumentation of the proof of Lemma 1 we have $c(D_x) \le 2 \cdot Opt(x)$. Since $m \le n$ for the length m of any path of D_x exchanged by a single edge, we obtain

$$c(H_x) \le (1+r)^{\lceil \log_2 n \rceil} \cdot c(D_x) \le 2 \cdot (1+r)^{\lceil \log_2 n \rceil} \cdot Opt(x) \ .$$

□

Now, we show that our approximation algorithms are really unstable according to *dist*. To show this, we construct an input, for which the Christofides algorithm provides a very poor approximation.

We construct a weighted complete graph from $Ball_{r,dist}(L_I)$ as follows. We start with the path $p_0, p_1, \ldots, p_n$ for $n = 2^k$, $k \in \mathbb{N}$, where every edge $\{p_i, p_{i+1}\}$ has the cost 1. Then we add edges (p_i, p_{i+2}) for $i = 0, 1, \ldots, n-2$ with the cost $2 \cdot (1+r)$. Generally, for every $m \in \{1, \ldots, \log_2 n\}$, we define $c(\{p_i, p_{i+2^m}\}) =$

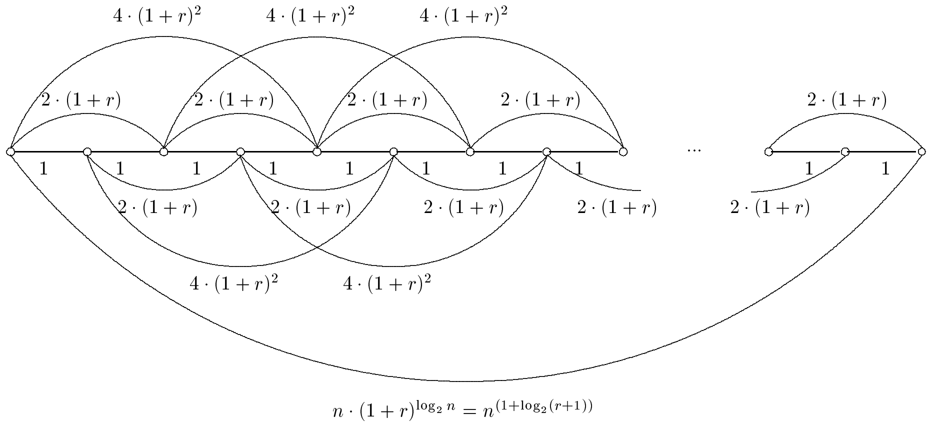

Fig. 1.

$2^m\cdot(1+r)^m$ for $i=0,\ldots,n-2^m$. For all other edges one can take the maximal possible cost in such a way that the constructed input is in $Ball_{r,dist}(L_I)$.

Let us have a look on the work of our algorithms on this input. There is only one minimal spanning tree that corresponds to the path containing all edges of the weight 1. Since every path contains exactly two nodes of odd degrees, the Euler tour constructed by the Christofides algorithm is the cycle $D=p_0,p_1,p_2,\ldots,p_n,p_0$ with the n edges of weight 1 and the edge of the maximal weight $n\cdot(1+r)^{\log_2 n}=n^{1+\log_2(1+r)}$. Since the Euler tour is a Hamiltonian tour, the output of the Christofides algorithm is unambigously the cycle $p_0,p_1,\ldots,p_n,p_0$ with the cost $n+n(1+r)^{\log_2 n}$. Observe, that the algorithm ST computes as output the same tour if the depth-first-search started from p_0 or from p_n. But the optimal tour is

$$T=p_0,p_2,p_4,\ldots,p_{2i},p_{2(i+1)},\ldots,p_n,p_{n-1},p_{n-3},\ldots,p_{2i+1},p_{2i-1},\ldots,p_3,p_1,p_0.$$

This tour contains two edges $\{p_0,p_1\}$ and $\{p_{n-1},p_n\}$ of the weight 1 and all $n-2$ edges of the weight $2\cdot(1+r)$. Thus, $Opt=cost(T)=2+2\cdot(1+r)\cdot(n-2)$, and

$$\begin{aligned}\frac{cost(D)}{cost(T)}&=\frac{n+n^{1+\log_2(1+r)}}{2+2\cdot(1+r)\cdot(n-2)}\\&\geq\frac{n^{1+\log_2(1+r)}}{2\cdot n\cdot(1+r)}=\frac{n^{\log_2(1+r)}}{2\cdot(1+r)}.\end{aligned}$$

So, we have proved the following result.

Lemma 3. *For every $r\in\mathbb{R}^+$, if the Christofides algorithm (the algorithm ST) is $(r,f(n,r))$-quasistable for dist, then $f(n,r)\geq n^{\log_2(1+r)}/2\cdot(1+r)$.*

Corollary 1. *The Christofides algorithm and the algorithm ST are unstable for dist.*

The results above show that the Christofides algorithm can be useful for a much larger set of inputs than the original input set. The key point is to find a suitable distance measure. In our case this measure is *distance* but not *dist*. An interesting question is whether one can modify these algorithms in such a way that the modified versions would be stable for *dist*.

Surprisingly, this is possible for both algorithms. To make the Christofides algorithm stable according to *dist* one first takes a minimal perfect path matching instead of an minimal perfect matching. In this way the cost of the Euler tour w remains at most $\frac{3}{2}\cdot$ the cost of an optimal Hamiltonian tour for any input instance of TSP. Secondly, we need a special procedure to produce a Hamiltonian tour H by shortening paths of w of the length at most 4 by an edge of H. Another possibility is to modify the ST algorithm. As we have observed in the example at Figure 1, the main problem is in the fact that shortening a path of a length m can increase the cost of this path by the multiplicative factor $(1+r)^{\log_2 m}$. To modify the ST algorithm one has to change the DFS-order of the vertices of the minimal spanning tree by another order, that shortens parts of the Euclid tour with length at most 3. There are too many technicalities to be able to explain these algorithms here into complete details. So, we present the final results achieved in the recent papers [1,5,6] only.

Theorem 1. *For every $\beta > 1$,*

(i) Δ_β-TSP can be approximated in polynomial-time within the approximation ratio $\min\left\{4\beta, \frac{3}{2}\beta^2\right\}$, *and*
(ii) unless P=NP, Δ_β-TSP cannot be approximated within approximation ratio $1+\varepsilon\cdot\beta$ for some $\varepsilon > 0$.

4 Superstability and Knapsack Problem

In this section we do not try to present any new result. We only use the known PTASs for the Knapsack problem (KP) to illustrate the transparency of the approximation stability point of view on their development. First we show that the original PTAS for the simple Knapsack problem (SKP) is stable, but not superstable, according a reasonable distance function. So, the application of this PTAS for KP leads to an approximation algorithm, but not to a PTAS. Then, we show that the known PTAS for KP is a simple modification of the PTAS for SKP, that makes this PTAS superstable according to our distance function.

The well-known PTAS for SKP works as follows:

PTAS SKP

Input: positive integers $w_1, w_2, \ldots, w_n, b$ for some $n \in \mathbb{N}$ and some positive real number $1 > \varepsilon > 0$.

Step 1: Sort $w_1, w_2, \ldots, w_n$. For simplicity we may assume $w_1 \geq w_2 \geq \cdots \geq w_n$.
Step 2: Set $k = \lceil 1/\varepsilon \rceil$.
Step 3: For every set $T = \{i_1, i_2, \ldots, i_l\} \subseteq \{1, 2, \ldots, n\}$ with $|T| = l \leq k$ and $\sum_{i \in T} w_i \leq b$, extend T to T' by using the greedy method and values $w_{i_l+1}, w_{i_l+2}, \ldots, w_n$. (The greedy method stops if $\sum_{i \in T'} w_i \leq b$ and $w_j > b - \sum_{i \in T'} w_i$ for all $j \notin T'$, $j \geq i_l$.)
Output: The best set T' constructed in Step 3.

For every given ε, we denote the above algorithm by A_ε. It is known that A_ε is an ε-approximation algorithm for SKP. Observe that it is consistent for KP. Now, we consider the following distance function *DIST* for any input $w_1, w_2, \ldots, w_n, b, c_1, \ldots, c_n$ of KP:

$$\mathit{DIST}(w_1, \ldots, w_n, b, c_1, \ldots, c_n) = \max\left\{\max\left\{\frac{c_i - w_i}{w_i} \,\middle|\, c_i \geq w_i,\ i \in \{1, \ldots, n\}\right\},\ \max\left\{\frac{w_i - c_i}{c_i} \,\middle|\, w_i \geq c_i,\ i \in \{1, \ldots, n\}\right\}\right\}.$$

Let $KP_\delta = (\Sigma_I, \Sigma_O, L, \mathit{Ball}_{\delta,\mathit{DIST}}(L_I), \mathcal{M}, \mathit{cost}, \mathit{maximum})$ for any δ. Now, we show that PTAS SKP is stable according to *DIST* but this result does not imply the existence of a PTAS for KP_δ for any $\delta > 0$.

Lemma 4. *For every $\varepsilon > 0$, and every $\delta > 0$, the algorithm A_ε is a $(1 + \varepsilon + \delta(2 + \delta) \cdot (1 + \varepsilon))$-approximation algorithm for KP_δ.*

Proof. Let $w_1 \geq w_2 \geq \cdots \geq w_n$ for an input $I = w_1, \ldots, w_n, b, c_1, \ldots, c_n$, and let $k = \lceil 1/\varepsilon \rceil$. Let $U = \{i_1, i_2, \ldots, i_l\} \subseteq \{1, 2, \ldots, n\}$ be an optimal solution for I. If $l \leq k$, then A_ε outputs an optimal solution with $\mathit{cost}(U)$ because A_ε has considered U as a candidate for the output in Step 3.

Consider the case $l > k$. A_ε has considered the greedy extension of $T = \{i_1, i_2, \ldots, i_k\}$ in Step 2. Let $T' = \{i_1, i_2, \ldots, i_k, j_{k+1}, \ldots, j_{k+r}\}$ be the greedy extension of T. Obviously, it is sufficient to show that the difference $\mathit{cost}(U) - \mathit{cost}(T')$ is small relative to $\mathit{cost}(U)$, because the cost of the output of A_ε is at least $\mathit{cost}(T')$. We distinguish the following two possibilities according to the weights of the feasable solutions U and T':

1. Let $\sum_{i \in U} w_i - \sum_{j \in T'} w_j \leq 0$.
   Obviously, for every i, $(1 + \delta)^{-1} \leq \frac{c_i}{w_i} \leq 1 + \delta$. So,
   $$\mathit{cost}(U) = \sum_{i \in U} c_i \leq (1 + \delta) \cdot \sum_{i \in U} w_i$$
   and
   $$\mathit{cost}(T') = \sum_{j \in T'} c_j \geq (1 + \delta)^{-1} \cdot \sum_{j \in T'} w_j \ .$$

In this way we obtain

$$\begin{aligned}
&\mathit{cost}(U) - \mathit{cost}(T') \leq \\
&\qquad \leq (1+\delta) \cdot \sum_{i \in U} w_i - (1+\delta)^{-1} \cdot \sum_{j \in T'} w_j \\
&\qquad \leq (1+\delta) \cdot \sum_{i \in U} w_i - (1+\delta)^{-1} \cdot \sum_{i \in U} w_i = \frac{\delta \cdot (2+\delta)}{1+\delta} \cdot \sum_{i \in U} w_i \\
&\qquad \leq \frac{\delta \cdot (2+\delta)}{1+\delta} \cdot \sum_{i \in U} (1+\delta) \cdot c_i = \delta \cdot (2+\delta) \cdot \sum_{i \in U} c_i \\
&\qquad\quad = \delta \cdot (2+\delta) \cdot \mathit{cost}(U).
\end{aligned}$$

Finally,

$$\frac{\mathit{cost}(U) - \mathit{cost}(T')}{\mathit{cost}(U)} \leq \frac{\delta \cdot (2+\delta) \cdot \mathit{cost}(U)}{\mathit{cost}(U)} = \delta \cdot (2+\delta).$$

2. Let $d = \sum_{i \in U} w_i - \sum_{j \in T'} w_j > 0$.
Let c be the cost of the first part of U with the weight $\sum_{j \in T'} w_j$. Then in the same way as in 1. one can establish

$$\frac{c - \mathit{cost}(T')}{c} \leq \delta \cdot (2+\delta)\ . \tag{1}$$

It remains to bound $\mathit{cost}(U) - c$, i.e. the cost of the last part of U with the weight d. Obviously, $d \leq b - \sum_{j \in T'} w_j \leq w_{i_r}$ for some $r > k$, $i_r \in U$ (if not, then A_ε would add r to T' in the greedy procedure). Since $w_{i_1} \geq w_{i_2} \geq \cdots \geq w_{i_l}$

$$d \leq w_{i_r} \leq \frac{w_{i_1} + w_{i_2} + \cdots + w_{i_r}}{r} \leq \frac{\sum_{i \in U} w_i}{k+1} \leq \varepsilon \cdot \sum_{i \in U} w_i \tag{2}$$

Since $\mathit{cost}(U) \leq c + d \cdot (1+\delta)$ we obtain

$$\begin{aligned}
\frac{\mathit{cost}(U) - \mathit{cost}(T')}{\mathit{cost}(U)} &\leq \frac{c + d \cdot (1+\delta) - \mathit{cost}(T')}{\mathit{cost}(U)} \\
&\overset{(2)}{\leq} \frac{c - \mathit{cost}(T')}{\mathit{cost}(U)} + \frac{(1+\delta) \cdot \varepsilon \cdot \sum_{i \in U} w_i}{\mathit{cost}(U)} \\
&\overset{(1)}{\leq} \delta \cdot (2+\delta) + (1+\delta) \cdot \varepsilon \cdot (1+\delta) \\
&= 2\delta + \delta^2 + \varepsilon \cdot (1+\delta)^2 \\
&= \varepsilon + \delta \cdot (2+\delta) \cdot (1+\varepsilon).
\end{aligned}$$

□

We see that the PTAS SKP is stable according to *DIST*, but this does not suffice to get a PTAS for KP_δ for any $\delta > 0$. This is because in the approximation ratio we have the additive factor $\delta \cdot (2+\delta)$ that is independent of ε. In what follows we change the PTAS SKP a little bit in such a way that we obtain a PTAS for every KP_δ, $\delta > 0$. The modification idea is very natural – to sort the input values according the cost per one weight unit.

PTAS MOD-SKP

Input: positive integers $w_1, w_2, \ldots, w_n, b, c_1, \ldots, c_n$ for some $n \in \mathbb{N}$ and some positive real number ε, $1 > \varepsilon > 0$.
Step 1: Sort $\frac{c_1}{w_1}, \frac{c_2}{w_2}, \ldots, \frac{c_n}{w_n}$. For simplicity we may assume $\frac{c_i}{w_i} \geq \frac{c_{i+1}}{w_{i+1}}$ for $i = 1, \ldots, n-1$.
Step 2: Set $k = \lceil 1/\varepsilon \rceil$.
Step 3: The same as Step 3 of PTAS SKP, but the greedy procedure follows the ordering of w_i's of Step 1.
Output: The best T' constructed in Step 3.

Let MOD-SK$_\varepsilon$ denote the algorithm given by PTAS MOD-SKP for a fixed $\varepsilon > 0$.

Lemma 5. *For every ε, $1 > \varepsilon > 0$ and every $\delta \geq 0$, MOD-SK$_\varepsilon$ is a $1+\varepsilon\cdot(1+\delta)^2$-approximation algorithm for SK$_\delta$.*

Proof. Let $U = \{i_1, i_2, \ldots, i_l\} \subseteq \{1, 2, \ldots, n\}$, where $w_{i_1} \leq w_{i_2} \leq \cdots \leq w_{i_l}$, be an optimal solution for the input $I = w_1, \ldots, w_n, b, c_1, \ldots, c_n$.

If $l \leq k$ then PTAS MOD-SKP$_\varepsilon$ provides an optimal solution.

If $l > k$, then we consider a $T' = \{i_1, i_2, \ldots, i_k, j_{k+1}, \ldots, j_{k+r}\}$ as a greedy extension of $T = \{i_1, i_2, \ldots, i_k\}$. Again, we distinguish two possibilities:

1. Let $\sum_{i \in U} w_i - \sum_{j \in T'} w_j < 0$.
   Now, we show that this contradicts to the optimality of U. Both, *cost*(U) and *cost*(T') contain $\sum_{s=1}^{k} c_{i_s}$. For the rest T' contains the best choice of w_i's according to the cost of one weight unit. The choice of U per one weight unit cannot be better. So, *cost*$(U) <$ *cost*(T').
2. Let $d = \sum_{i \in U} w_i - \sum_{j \in T'} w_j \geq 0$.
   Because of the optimal choice of T' according to the cost per one weight unit, the cost c of the first part of U with the weight $\sum_{j \in T'} w_j$ is at most *cost*(T'), i.e.,

$$c - \mathit{cost}(T') \leq 0 \ . \tag{3}$$

   Since U and T' contain the same k indices $i_1, i_2, \ldots, i_k$ and $w_{i_1}, \ldots, w_{i_k}$ are the largest weights in both U and T', the same same consideration as in the proof of Lemma 1 (see (2)) yields

$$d \leq \varepsilon \cdot \sum_{i \in U} w_i, \text{ and } \mathit{cost}(U) \leq c + d \cdot (1+\delta) \ . \tag{4}$$

Thus,

$$\begin{aligned}\frac{cost(U)-cost(T')}{cost(U)} &\leq \frac{c+d\cdot(1+\delta)-cost(T')}{cost(U)}\\ &\overset{(3)}{\leq} \frac{d\cdot(1+\delta)}{cost(U)} \overset{(4)}{\leq} \varepsilon\cdot(1+\delta)\cdot\frac{\sum_{i\in U} w_i}{cost(U)} = \varepsilon\cdot(1+\delta)^2 \ .\end{aligned}$$

□

We observe that the collection of MOD-KP$_\varepsilon$ algorithms is a PTAS for every KP_δ with a constant $\delta \geq 0$ (independent of the size $2n+1$ of the input).

5 Conclusion and Discussion

In the previous sections we have introduced the concept of stability of approximations. Here we discuss the potential applicability and usefulness of this concept. Using this concept, one can establish positive results of the following types:

1. An approximation algorithm or a PTAS can be successfully used for a larger set of inputs than the set usually considered (see Lemma 1 and Lemma 4).
2. We are not able to successfully apply a given approximation algorithm A (a PTAS) for additional inputs, but one can simply modify A to get a new approximation algorithm (a new PTAS) working for a larger set of inputs than the set of inputs of A.
3. To learn that an approximation algorithm is unstable for a distance measure could lead to the development of completely new approximation algorithms that would be stable according to the considered distance measure.

The following types of negative results may be achieved:

4. The fact that an approximation algorithm is unstable according to all "reasonable" distance measures and so that its use is really restricted to the original input set.
5. Let $Q = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal) \in NPO$ be well approximable. If, for a distance measure D and a constant r, one proves the nonexistence of any polynomial-time approximation algorithm for $Q_{r,D} = (\Sigma_I, \Sigma_O, L, Ball_{r,D}(L_I), \mathcal{M}, cost, goal)$, then this means that the problem Q ist "unstable" according to D.

Thus, using the notion of stability one can search for a spectrum of the hardness of a problem according to the set of inputs. For instance, considering a hard problem like TSP one could get an infinite sequence of input languages $L_0, L_1, L_2, \ldots$ given by some distance measure, where $\varepsilon_r(n)$ is the best achievable relative error for the language L_r. Results of this kind can essentially contribute to the study of the nature of hardness of specific computing problems.

References

1. T. Andreae, H.-J. Bandelt: Performance guarantees for approximation algorithms depending on parametrized triangle inequalities. *SIAM J. Disr. Math.* 8 (1995), pp. 1–16. 41
2. S. Arora: Polynomial time approximation schemes for Euclidean TSP and other geometric problems. In: *Proc. 37th IEEE FOCS*, IEEE 1996, pp. 2–11. 32
3. S. Arora: Nearly linear time approximation schemes for Euclidean TSP and other geometric problems. In: *Proc. 38th IEEE FOCS*, IEEE 1997, pp. 554–563. 32
4. D. P. Bovet, C. Crescenzi: *Introduction to the Theory of Complexity*, Prentice-Hall, 1993. 31, 33
5. M. A. Bender, C. Chekuri: Performance guarantees for the TSP with a parameterized triangle inequality. In: *Proc. WADS'99, LNCS*, to appear. 41
6. H.-J. Böckenhauer, J. Hromkovič, R. Klasing, S. Seibert, W. Unger: Towards the Notion of Stability of Approximation Algorithms and the Traveling Salesman Problem. Unpublished manuscript, RWTH Aachen, 1998. 41
7. 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, Pittsbourgh, 1976. 31, 31
8. P. Crescenzi, V. Kann: *A compendium of NP optimization problems.* http://www.nada.kth.se/theory/compendium/. 34
9. S. A. Cook: The complexity of theorem proving procedures. In: Proc *3rd ACM STOC*, ACM 1971, pp. 151–158. 31, 31
10. M. R. Garey, D. S. Johnson: *Computers and Intractibility. A Guide to the Theory on NP-Completeness.* W. H. Freeman and Company, 1979. 33
11. K. Gödel: Über formal unentscheidbare Sätze der Principia Mathematica und verwandte Systeme, I. *Monatshefte für Mathematik und Physik* 38 (1931), pp. 173–198. 29
12. J. Håstad: Some optimal inapproximability results. In: *Proc. 29th ACM STOC*, ACM 1997, pp. 1–10.
13. D. S. Hochbaum (Ed.): *Approximation Algorithms for NP-hard Problems.* PWS Publishing Company 1996. 33, 34, 35
14. O. H. Ibarra, C. E. Kim: Fast approximation algorithms for the knapsack and sum of subsets problem. *J. of the ACM* 22 (1975), pp. 463–468. 31, 31
15. D. S. Johnson: Approximation algorithms for combinatorial problems *JCSS* 9 (1974), pp. 256–278. 31
16. R. M. Karp: Reducibility among combinatorial problems. In: R. E. Miller, J. W. Thatcher (Eds.): *Complexity of Computer Computations*, Plenum Press 1972, pp. 85–103. 31
17. L. Lovász: On the ratio of the optimal integral and functional covers. *Discrete Mathematics* 13 (1975), pp. 383–390. 31
18. I. S. B. Mitchell: Guillotine subdivisions approximate polygonal subdivisions: Part II – a simple polynomial-time approximation scheme for geometric k-MST, TSP and related problems. Technical Report, Dept. of Applied Mathematics and Statistics, Stony Brook 1996. 32
19. E. W. Mayr, H. J. Prömel, A. Steger (Eds.): *Lecture on Proof Verification and Approximation Algorithms. Lecture Notes in Computer Science* 1967, Springer 1998. 31, 35
20. Ch. Papadimitriou: The Euclidean travelling salesman problem is NP-complete. *Theoretical Computer Science* 4 (1977), pp. 237–244. 32

21. Ch. Papadimitriou: *Computational Complexity*, Addison-Wesley 1994. 31, 33
22. B. A. Trachtenbrot: *Algorithms and Automatic Computing Machines.* D. C. Heath & Co., Boston, 1963. 30
23. I. Wegener: *Theoretische Informatik: eine algorithmenorientierte Einführung.* B. G. Teubner 1993. 33