Electronic Colloquium on Computational Complexity, Report No. 31 (1999)

ECCC

Towards the Notion of Stability of Approximation for Hard Optimization Tasks and the Traveling Salesman Problem

Hans-Joachim Böckenhauer Juraj Hromkovič Ralf Klasing
Sebastian Seibert Walter Unger

{hjb,jh,rak,seibert,quax}@i1.informatik.rwth-aachen.de
Lehrstuhl für Informatik I (Algorithmen und Komplexität)
RWTH Aachen, 52056 Aachen, Germany
Fax: ++49 +241 - 8888 216

September 2, 1999

Abstract

The investigation of the possibility to efficiently compute approximations of hard optimization problems is one of the central and most fruitful areas of current algorithm and complexity theory. The aim of this paper is twofold. First, we introduce the notion of stability of approximation algorithms. This notion is shown to be of practical as well as of theoretical importance, especially for the real understanding of the applicability of approximation algorithms and for the determination of the border between easy instances and hard instances of optimization problems that do not admit polynomial-time approximation.

Secondly, we apply our concept to the study of the traveling salesman problem. We show how to modify the Christofides algorithm for Δ-TSP to obtain efficient approximation algorithms with constant approximation ratio for every instance of TSP that violates the triangle inequality by a multiplicative constant factor. This improves the result of Andreae and Bandelt [AB95].

Keywords: Stability of approximation, Traveling Salesman Problem

1 Introduction

Immediately after introducing NP-hardness (completeness) [Co71] as a concept for proving intractability of computing problems, the following question has been posed: If an optimization problem does not admit an efficiently computable optimal solution, is there a possibility to efficiently compute at least an approximation of the optimal solution? Several researchers [Jo74], [Lo75], [Chr76], [IK75] provided already in the middle of the seventies a positive answer for some optimization problems. It is a fascinating effect if one can jump from exponential complexity (a huge inevitable amount of physical work) to polynomial complexity (tractable amount of physical

ISSN 1433-8092

work) due to a small change in the requirements — instead of an exact optimal solution one demands a solution whose cost differs from the cost of an optimal solution by at most $\varepsilon\%$ of the cost of an optimal solution for some $\varepsilon > 0$. This effect is very strong, especially, if one considers problems for which this approximation concept works for any relative difference ε (see the concept of approximation schemes in [IK75], [MPS98], [Pa94], [BC93]). This is also the reason 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 any 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 Salesman Problem (TSP). TSP is not only NP-hard, but also the search for an approximate solution for TSP is NP-hard for every constant approximation ratio.[1] But if one considers TSP for inputs satisfying the triangle inequality (so called Δ-TSP), one can even design a polynomial-time $\frac{3}{2}$-approximation algorithm [Chr76]. The situation is even more 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 [Pa77], but for every $\alpha > 1$ one can design a polynomial-time α-approximation algorithm [Ar96], [Mi96]. Moreover, if one allows randomization the resulting approximation algorithm works in $n \cdot (\log_2 n)^{O(1)}$ time [Ar97].[2] This is the reason why we propose again to revise the notion of tractability especially because of the standard definition of complexity as the worst-case complexity: Our aim is to try to separate the easy instances from the hard instances of every computing problem considered to be intractable. In fact, by our concept, we want to attack the definition of complexity as the worst-case complexity. The approximation ratio of an algorithm is also defined in a worst-case manner. Our idea is to split the set of input instances of the given problem into possibly infinitely many subclasses according to the hardness of their approximability, and to have an efficient algorithm for deciding the membership of any problem instance to one of the subclasses considered. 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 for two sets of inputs L_1 and L_2, $L_1 \subsetneq L_2$. For L_1 there exists a polynomial-time δ-approximation algorithm A for some $\delta > 1$, but for L_2 there is no polynomial-time γ-approximation algorithm for any $\gamma > 1$ (if NP is not equal to P). We pose the following question: Is the use of algorithm A really restricted to inputs from L_1? Let us consider a distance measure d in L_2 determining the distance $d(x)$ between L_1 and any given input $x \in L_2 - L_1$. Now, one can consider an input $x \in L_2 - L_1$ with $d(x) \leqslant k$ 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 $d(x) \leqslant k$, A computes a $\gamma_{k,\delta}$-approximation of

[1] Even no $f(n)$-approximation algorithm exists for f exponential in the input size n.

[2] Obviously, there are many 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).

an optimal solution for x ($\gamma_{k,\delta}$ is considered to be a constant depending on k and δ only), then one can say that A is "(approximation) stable" according to the distance measure d. Obviously, such a concept enables to show positive results extending the applicability of known approximation algorithms. On the other hand it can help to show the boundaries of the use of approximation algorithms and possibly even a new kind of hardness of optimization problems.

Observe that the idea of the concept of approximation stability is similar to that of stability of numerical algorithms. Instead of observing the size of the change of the output value according to a small change of the input value, one looks for the size of the change of the approximation ratio according to a small change in the specification of the set of consistent input instances.

To demonstrate the applicability of our new approach we consider TSP, Δ-TSP, and, for every real $\beta > 1$, Δ_β-TSP containing all input instances with $cost(u,v) \leqslant \beta \cdot (cost(u,x) + cost(x,v))$ for all vertices u, v, x. If an input is consistent for Δ_β-TSP we say that its distance to Δ-TSP is at most $\beta - 1$. We will show that known approximation algorithms for Δ-TSP are unstable according to this distance measure. But we will find a way how to modify the Christofides algorithm in order to obtain approximation algorithms for Δ-TSP that are stable according to this distance measure. So, this effort results in a $(\frac{3}{2} \cdot \beta^2)$-approximation algorithm for Δ_β-TSP.[3] This improves the result of Andreae and Bandelt [AB95] who presented a $(\frac{3}{2}\beta^2 + \frac{1}{2}\beta)$-approximation algorithm for Δ_β-TSP. Our approach essentially differs from that of [AB95], because in order to design our $(\frac{3}{2} \cdot \beta^2)$-approximation algorithm we modify the Christofides algorithm while Andreae and Bandelt obtain their approximation ratio by modifying the original 2-approximation algorithm for Δ-TSP.

Note that, after this paper was written, we got the information about the independent, unpublished result of Bender and Chekuri, accepted for WADS'99 [BC99]. They designed a 4β-approximation algorithm which can be seen as a modification of the 2-approximation algorithm for Δ_β-TSP. Despite this nice result, there are three reasons to consider our algorithm. First, our algorithm provides a better approximation ratio for $\beta < \frac{8}{3}$. Secondly, in the previous work [AB95], the authors claim that the Christofides algorithm cannot be modified in order to get a stable (in our terminology) algorithm for TSP, and our result disproves this conjecture. This is especially of practical importance, since for instances where the triangle inequality is violated only by a few edge costs, one can expect that the approximation ratio will be as in the underlying algorithm with a high probability. Finally, our algorithm is a practical $O(n^3)$-algorithm. This cannot be said about the 4β-approximation algorithm from [BC99]. The first part of the latter algorithm is a 2-approximation algorithm for finding minimal two-connected subgraphs with time complexity $O(n^4)$. For the second part, constructing a Hamiltonian tour in S^2 (if S was the two-connected subgraph), there exist only proofs saying that it can be implemented in polynomial time, but no low-degree polynomial upper bound on the time complexity of these procedures has been established.

This paper is organized as follows: In Section 2 we introduce our concept of

[3]Note that in this way we obtain an approximate solution to every problem instance of TSP, where the approximation ratio depends on the distance of this problem instance to Δ-TSP. Following the discussion in [Ar96] about typical properties of real problem instances of TSP our approximation algorithm working in $O(n^3)$ time is of practical relevance.

approximation stability. In Section 3 we show how to apply our concept in the study of the TSP, and in Section 4 we discuss the potential applicability and usefulness of our concept.

2 Definition of the Stability of Approximation Algorithms

We assume that the reader is familiar with the basic concepts and notions of algorithmics and complexity theory as presented in standard textbooks like [BC93], [CLR90], [GJ79], [Ho96], [Pa94]. Next, we give a new 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, let $\mathbb{R}^+$ be the set of positive reals, and let $\mathbb{R}^{\geq a}$ be the set of all reals greater than or equal to a for some $a \in \mathbb{R}$.

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

1. Σ_I is an alphabet called **input alphabet**,
2. Σ_O is an alphabet called **output alphabet**,
3. $L \subseteq \Sigma_I^*$ is a language over Σ_I called the **language of consistent inputs**,
4. $L_I \subseteq L$ is a language over Σ_I called the **language of actual inputs**,
5. $\mathcal{M}$ 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,
6. $cost$ is a function, called **cost function**, from $\bigcup_{x \in L} \mathcal{M}(x) \times L_I$ to $\mathbb{R}^{\geq 0}$,
7. $goal \in \{minimum, maximum\}$.

For every $x \in L$, we define

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

and

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

□

Clearly, the meaning for Σ_I, Σ_O, $\mathcal{M}$, $cost$ and $goal$ is the usual one. L may be considered as the 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 parameterized according to their languages of actual inputs. In what follows, $\boldsymbol{Language(U)}$ denotes the language L_I of actual inputs of U. If the input x is fixed, we usually use $cost(y)$ instead of $cost(y, x)$ in what follows.

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 is defined as the function

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

from $\mathbb{N}$ to $\mathbb{N}$, where $\boldsymbol{Time_A(x)}$ 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. [CK98], [Ho96]).

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 **approximation ratio $R_A(x)$** is defined as

$$\boldsymbol{R_A(x)} = \max\left\{\frac{cost(A(x))}{Opt_U(x)}, \frac{Opt_U(x)}{cost(A(x))}\right\}.$$

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

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

For any positive real $\delta > 1$, we say that A is a **δ-approximation algorithm for U** if $R_A(x) \leqslant \delta$ for every $x \in L_I$.
For every function $f : \mathbb{N} \to \mathbb{R}^{>1}$, we say that A is an **$f(n)$-approximation algorithm for U** if $R_A(n) \leqslant f(n)$ for every $n \in \mathbb{N}$.

□

In what follows, we consider the standard definitions of the classes NPO, PO, APX (see e.g. [Ho96],[MPS98]). 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 4 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 $L_I \subsetneq L$. A **distance function for U according to L_I** is any function $h_L : L \to \mathbb{R}^{\geqslant 0}$ satisfying the properties

1. $h_L(x) = 0$ for every $x \in L_I$, and
2. h_L can be computed in polynomial time.

Let h be a distance function for U according to L_I. We define, for any $r \in \mathbb{R}^+$,

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

Let A be a consistent algorithm for $\overline{U}$, and let A be an ε-approximation algorithm for U for some $\varepsilon \in \mathbb{R}^{>1}$. Let p be a positive real. We say that A is **p-stable**

according to h if, for every real $0 < r \leqslant p$, there exists a $\delta_{r,\varepsilon} \in \mathbb{R}^{>1}$ such that A is a $\delta_{r,\varepsilon}$-approximation algorithm for $U_r = (\Sigma_I, \Sigma_O, L, Ball_{r,h}(L_I), \mathcal{M}, cost, goal)$.[4]

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 $f_r : \mathbb{N} \to \mathbb{R}^{>1}$ we say that A is **$(r, f_r(n))$-quasistable according to h** if A is an $f_r(n)$-approximation algorithm for $U_r = (\Sigma_I, \Sigma_O, L, Ball_{r,h}(L_I), \mathcal{M}, cost, goal)$.

□

A discussion about the potential usefulness of our concept is given in the last section. In the next section we show a transparent application of our concept for TSP.

3 Stability of Approximation Algorithms and TSP

We consider the well-known TSP problem (see e.g. [LLRS85]) 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 a $\frac{3}{2}$-approximation algorithm due to Christofides [Chr76]. So, this is a suitable starting point for the application of our approach based on approximation stability. First, we define two natural distance measures and show that the Christofides algorithm is stable according to one of them, but not according to the second one. This leads to the development of a new algorithm, PMCA, for Δ_β-TSP. This algorithm is achieved by modifying Christofides algorithm in such a way that the resulting algorithm is stable according to the second distance measure, too. In this way, we obtain a $(\frac{3}{2}\cdot(1+r)^2)$-approximation algorithm for every input instance of TSP with the distance at most r from $Language(\Delta$-TSP), i.e. with $cost(u,v) \leqslant (1+r)\cdot(cost(u,w)+cost(w,v))$ for every three nodes u, v, w. This improves the result of Andreae and Bandelt [AB95] who achieved approximation ratio $\frac{3}{2}(1+r)^2 + \frac{1}{2}(1+r)$.

To start our investigation, we concisely review two well-known algorithms for Δ-TSP: the 2-approximative algorithm 2APPR and the $\frac{3}{2}$-approximative Christofides algorithm [Chr76], [Ho96].

Algorithm 2APPR

Input: A complete graph $G = (V, E)$ with a cost function $cost : E \to \mathbb{R}^{\geqslant 0}$ satisfying the triangle inequality (for every $u, v, q \in V$, $cost(u,v) \leqslant cost(u,q) + cost(q,v)$).

Step 1: Construct a minimal spanning tree T of G. (The cost of T is surely smaller than the cost of the optimal Hamiltonian tour.)

Step 2: Construct a Eulerian tour D on T going twice via every edge of T. (The cost of D is exactly twice the cost of T.)

Step 3: Construct a Hamiltonian tour H from D by avoiding the repetition of nodes in the Eulerian tour. (In fact, H is the permutation of nodes of G, where the order of a node v is given by the first occurrence of v in D.)

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

Output: H.

Christofides Algorithm

Input: A complete graph $G = (V, E)$ with a cost function $cost : E \to \mathbb{R}^{\geqslant 0}$ satisfying the triangle inequality.

Step 1: Construct a minimal spanning tree T of G and find a matching M with minimal cost (at most $\frac{1}{2}$ of the cost of the optimal Hamiltonian tour) on the nodes of T with odd degree.

Step 2: Construct a Eulerian tour D on $G' = T \cup M$.

Step 3: Construct a Hamiltonian tour H from D by avoiding the repetition of nodes in the Eulerian tour.

Output: H.

Since the triangle inequality holds and Step 3 in both algorithms is realized by repeatedly shortening a path $x, u_1, \ldots, u_m, y$ by the edge (x, y) (because $u_1, \ldots, u_m$ have already occurred before in the prefix of D) the cost of H is at most the cost of D. Thus, the crucial point for the success of 2APPR and Christofides algorithm is the triangle inequality. A reasonable possibility to search for an extension of the application of these algorithms is to look for inputs that "almost" satisfy the triangle inequality. In what follows we do this in two different ways.

Let Δ-TSP $= (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, minimum)$ be a representation of the TSP with triangle inequality. We may assume $\Sigma_I = \Sigma_O = \{0, 1, \#\}$, L contains codes of all cost functions for edges of complete graphs, and L_I contains codes of cost functions that satisfy the triangle inequality. Let, for every $x \in L$, $G_x = (V_x, E_x, cost_x)$ be the complete weighted graph coded by x. Obviously, the Christofides algorithm is consistent for $(\Sigma_I, \Sigma_O, L, L, \mathcal{M}, cost, minimum)$.

Let $\Delta_{1+r,d}$-TSP $= (\Sigma_I, \Sigma_O, L,\ Ball_{r,d}(L_I),\ \mathcal{M}, cost, minimum)$ for any $r \in \mathbb{R}^+$ and for any distance function d for Δ-TSP. We define for every $x \in L$,

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

and

$$\begin{aligned} distance(x) \quad = \quad & \max\left\{0, \max\left\{\frac{cost(u,v)}{\sum_{i=1}^{m} cost(p_i, p_{i+1})} - 1 \,\middle|\, u, v \in V_x,\right.\right. \\ & \text{and } u = p_1, p_2, \ldots, p_{m+1} = v \text{ is a simple path between} \\ & \left.\left. u \text{ and } v \text{ in } G_x\right\}\right\}. \end{aligned}$$

Since the distance measure *dist* is the most important for us we will use the notation Δ_β-TSP instead of $\Delta_{\beta,dist}$-TSP. For simplicity we consider the size of x as the number of nodes of G_x instead of $|x|$. We observe that for every $\beta \geqslant 1$ the inputs from $\Delta_{\beta,dist}$-TSP have the property $cost(u, v) \leqslant \beta \cdot (cost(u, x) + cost(x, v))$ for all u, v, x $(\beta = 1 + r)$.

Lemma 1 *The 2APPR and Christofides algorithm are stable according to distance.*

Proof. We present the proof for the algorithm 2APPR 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 $H_x = v_1, v_2, ..., v_n, v_1$ be the resulting Hamiltonian tour. Obviously, D_x can be written as $v_1 P_1 v_2 P_2 v_3 ... v_n P_n v_1$, 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 $i \in \{1, 2, ..., n\}$, 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 2APPR is a $(2 \cdot (1+r))$-approximation algorithm for $(\Sigma_I, \Sigma_O, L, Ball_{r,distance}(L_I), \mathcal{M}, cost, minimum)$. □

Now, one can ask for the approximation stability according to the distance measure *dist* that is the most interesting distance measure for us. Unfortunately, as shown in the next lemmas, the answer is not as positive as for *distance*.

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

Proof. Again, we realize the proof for the algorithm 2APPR only. Let $x = (G, c) \in Ball_{r,dist}(L_I)$ 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 $(p, s), (s, t)$ 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

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

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

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

□

That the result of Lemma 2 cannot be essentially improved is shown by presenting an input for which the Christofides algorithm as well as 2APPR provide a very poor approximation.

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

Proof. 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. For all other edges we take maximal possible costs in such a way that the constructed input is in $Ball_{r,dist}(L_I)$. As a consequence, for every $m \in \{1, \ldots, \log_2 n\}$, we have $cost(p_i, p_{i+2^m}) = 2^m \cdot (1+r)^m$ for $i = 0, \ldots, n - 2^m$ (see Figure 1).

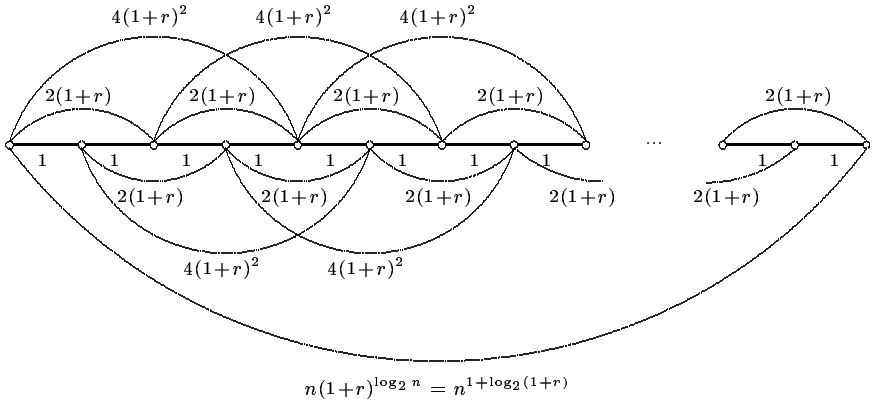

Figure 1: A hard $\Delta_{\beta,dist}$-TSP instance

Let us have a look at the work of Christofides algorithm on this input. (Similar considerations can be made for 2APPR.) There is only one minimal spanning tree that corresponds to the path containing all edges of the cost 1. Since every path contains exactly two nodes of odd degrees, the Eulerian graph constructed in Step 1 is the cycle $D = p_0, p_1, p_2, \ldots, p_n, p_0$ with the n edges of cost 1 and the edge of the maximal cost $n \cdot (1+r)^{\log_2 n} = n^{1+\log_2(1+r)}$. Since the Eulerian path is a Hamiltonian tour, the output of the Christofides algorithm is unambiguously the cycle $p_0, p_1, \ldots, p_n, p_0$ with the cost $n + n^{1+\log_2(1+r)}$. But the optimal tour is

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

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

$$\frac{cost(D)}{cost(H)} = \frac{n + n^{1+\log_2(1+r)}}{2 + 2 \cdot (1+r) \cdot (n-2)} \geqslant \frac{n^{1+\log_2(1+r)}}{2 \cdot n \cdot (1+r)} = \frac{n^{\log_2(1+r)}}{2 \cdot (1+r)}.$$

□

Corollary 1 *2APPR and the Christofides algorithm are unstable for dist.* □

The results above show that 2APPR and Christofides algorithm can be useful for a much larger set of inputs than the original input set. But the stability according to *dist* would provide approximation algorithms for a substantially larger class of input instances. So the key question is whether one can modify the above algorithms to get algorithms that are stable according to *dist*. In what follows, we give a positive answer on this question.

4 A Stable Algorithm for $\Delta_{\beta,dist}$-TSP

The main aim of this section is to prove the following result.

Theorem 1 *For every $\beta \in \mathbb{R}^{\geq 1}$, there is a $(\frac{3}{2} \cdot \beta^2)$-approximation algorithm PMCA for $\Delta_{\beta,dist}$-TSP working in time $O(n^3)$.*

The informal idea of the algorithm PMCA is as follows. The first step is a modification of the Christofides algorithm. Instead of building G' from the minimal spanning tree T and a matching on nodes with odd degrees we build G' from T and a "path matching" consisting of cheapest paths between the nodes of T with odd degrees. Using this we avoid to take edges with high costs, and the cost of our resulting graph G' (and also of the corresponding Eulerian tour D) will be again (as in the case of Δ-TSP) at most $\frac{3}{2}$ times the cost of an optimal Hamiltonian tour.

The main part of PMCA deals with the task of transforming D into a Hamiltonian tour by substituting at most 4 consecutive edges by a new one. Under the given modified triangle inequality, this guarantees that the new tour costs at most β^2 as much as D which gives the desired quality.

The way how PMCA realizes this task is to resolve conflicts (vertices multiply used in D) in three steps. First, all conflicts within the path matching are resolved even before constructing the Eulerian tour D. Next, after D is obtained, by resolving some conflicts the spanning tree is divided into a forest of degree at most 3. The preceding steps will assure that, in every conflict resolution, at most two consecutive edges are substituted by a new one. Moreover these new edges are separated in some way, such that in the final step at most one of them together with two original edges will be substituted by another new one when PMCA resolves all remaining conflicts on the cycle.

The rather technical proof of Theorem 1 follows. Theorem 1 improves the approximation ratio achieved in [AB95]. Note that this cannot be done by modifying the approach of Andreae and Bandelt. The crucial point of our improvement is based on the presented modification of Christofides algorithm while Andreae and Bandelt conjectured in [AB95] that Christofides algorithm cannot be modified in order to get an approximation algorithm for $\Delta_{\beta,dist}$-TSP.

Note that Theorem 1 can also be formulated in a general form by substituting the parameter β by a function $\beta(n)$, where n is the number of nodes of the graph considered.

Proof. of Theorem 1.

In the following, we will give a proof of Theorem 1 by stating algorithm PMCA and showing its approximation ratio and time complexity. The central ideas of PMCA are the following. First, we replace the minimum matching generated in the Christofides Algorithm by a "minimum path matching". That means to find a pairing of the given vertices s.t. the vertices in a pair are connected by a path rather than a single edge, and the goal is to minimize the sum of the path costs. In this way, we obtain an Eulerian tour on the multi-graph consisting of spanning tree and path matching (in general not Hamiltonian). This Eulerian tour has a cost of at most 1.5 times of the cost of an optimal TSP tour, as will be shown in Claim 7.

The second new concept concerns the substitution of sequences of edges by single ones, when transforming the above mentioned tour to a Hamiltonian one. Here, we can guarantee that at most four consecutive edges will be eventually substituted by a single one. This may increase the cost of the tour by a factor of at most β^2 for inputs from Δ_β-TSP (remember that we deal in this section only with distance function *dist*, and therefore drop the corresponding subscript from $\Delta_{\beta,dist}$-TSP).

Before stating in detail the algorithm, we have to introduce its main tools first. Let $G = (V, E)$ be a graph. A *path matching* for a set of vertices $U \subseteq V$ of even size is a collection Π of $|U|/2$ edge-disjoint paths having U as the set of endpoints.

Assume that $p = (u_0, u_1), (u_1, u_2), \ldots, (u_{k-1}, u_k)$ is a path in (V, E), not necessarily simple. A *bypass* for p is an edge (u, v) from E, replacing a sub-path $(u_i, u_{i+1}), (u_{i+1}, u_{i+2}), \ldots, (u_{j-1}, u_j)$ of p from $u = u_i$ to $u_j = v$ $(0 \leqslant i < j \leqslant k)$. Its *size* is the number of replaced edges, i.e. $j - i$.[5] Also, we say that the vertices $u_{i+1}, u_{i+2}, \ldots, u_{j-1}$ *are bypassed*. Given some set of simple paths Π, a *conflict* according to Π is a vertex which occurs at least two times in the given set of paths.

Algorithm PMCA

Input: a complete graph (V, E) with cost function $cost : E \to \mathbb{R}^{\geqslant 0}$
(a Δ_β-TSP instance for $\beta \geq 1$).
1. Construct a minimal spanning tree T of (V, E).
2. Let U be the set of vertices of odd degree in T;
construct a minimal (edge-disjoint) path matching Π for U.
3. Resolve conflicts according to Π, in order to
obtain a vertex-disjoint path matching Π' with $cost(\Pi') \leqslant \beta \cdot cost(\Pi)$
(using bypasses of size 2 only).
4. Construct an Eulerian tour π on T and Π'.
(π can be considered as a sequence of paths $p_1, p_2, p_3, \ldots$
such that $p_1, p_3, \ldots$ are paths in T, and $p_2, p_4, \ldots \in \Pi'$)
5. Resolve conflicts inside the paths $p_1, p_3, \ldots$ from T, such that T is divided into
a forest T_f of trees of degree at most 3, using bypasses of size 2 only.
(Call the resulting paths $p'_1, p'_3, \ldots$ and the modified tour π' is $p'_1, p_2, p'_3, p_4, \ldots$.)
6. Resolve every double occurrence of nodes in π' such that the overall size
of the bypasses is at most 4 (where "overall" means that a bypass constructed
in Step 3 counts for two edges). Obtain tour π''.
Output: Tour π''.

In the following, we have to explain how to efficiently obtain a minimal path matching, and how to realize the conflict resolution in Steps 3, 5, and 6. The latter not only have to be efficient but must also result in substituting at most four edges by a single one after all.

How to construct an Eulerian cycle in Step 4 is a well-studied task. We only observe that since each vertex can be endpoint of at most one path from Π' by definition the same holds for the paths in T: the endpoints of $p_1, p_3, \ldots$ are the same as those of $p_2, p_4, \ldots$.

[5] Obviously, we are not interested in bypasses of size 1.

Remark 1
In the Eulerian cycle constructed in Step 4 of Algorithm PMCA, every vertex occurs at most once as endpoint of a path built from the spanning tree. □

We give in Sections 4.1, 4.2, 4.3, and 4.4 detailed descriptions of Steps 2, 3, 5, and 6, respectively. Finally, in Section 4.5, we show that Algorithm PMCA meets the claimed performance guarantee.

4.1 Computing minimum path matchings

Claim 1
One can construct in time $O(|V|^3)$ a minimum path matching Π for U that has the following properties:

Every two paths in Π are edge-disjoint. (1)

Π forms a forest. (2)

Proof. First, we will show how to construct a path matching within the given time. Then we will show that the claimed properties (1) and (2) are obtained as a consequence of the minimality. For showing this, we will assume initially that there are no edges of cost zero, and at the end, we will demonstrate how to cope with such edges by applying a technical trick.

To construct the path matching, we first compute all-pairs cheapest paths.[6] Then, we define $G' = (V, E')$ where $cost'(v, v')$ is the cost of a cheapest path between v and v' in G. Next, we compute a minimum matching on G' (in the usual sense), and finally, we substitute the edges of G' in the matching by the corresponding cheapest paths in G. Clearly, this can be done in time $O(n^3)$ and results in a minimum path matching.

Assume now that there are two paths having one or more common edges as shown in Figure 2(a). Substituting them by paths as in Figure 2(b) removes the common edges from the path matching and anything in between. Due to the minimality, the removed part consisted of zero-cost edges.

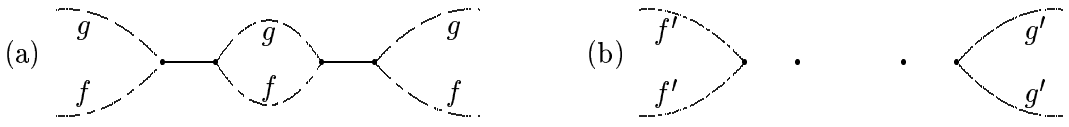

Figure 2: Impossibility of multiply used edges in a minimal path matching

Similarly, if the path matching is not a forest, it contains cycles as in Figure 3(a). These can be removed as shown in Figure 3(b). By the same argument as above, the removed edges must be of cost zero.

[6]Since we associate a cost instead of a length to the edges, we speak about cheapest instead of shortest path.

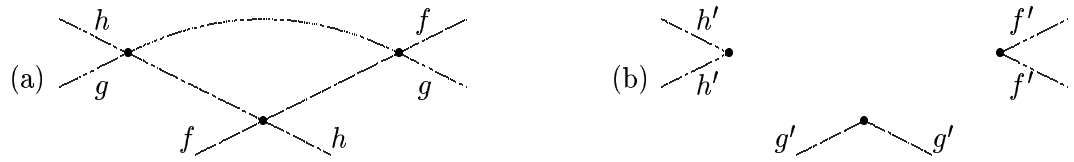

Figure 3: Impossibility of cycles in a minimal path matching

The above considerations show that in the absence of zero-cost edges, any minimal path matching will have the claimed properties.

If there are zero-cost edges present in the graph, we compute the path matching by using the following modified cost function:

$$\widehat{cost}(u,v) = (cost(u,v), 1)$$

for each edge (u,v). Addition is done component-wise, and costs are ordered by

$$(c_1,c_2) < (d_1,d_2) \text{ iff } \left(\begin{array}{l} c_1 < d_1, \text{ or} \\ c_1 = d_1 \text{ and } c_2 < d_2. \end{array} \right)$$

It is clear that a minimum path matching according to the new cost function is minimal according to the original one, too. Since the new function avoids zero cost, the claimed properties are guaranteed. □

The essential property of a minimal path matching for our purposes is that it costs at most half of the cost of a minimal Hamiltonian tour. This will be shown in Claim 7 in Section 4.5 where we estimate the approximation ratio of Algorithm PMCA.

4.2 Conflict elimination in the path matching Π

Now we show how Step 3 of the algorithm is performed. That is, we will show the following claim.

Claim 2
Every path matching having properties (1) and (2) can be modified into a vertex-disjoint one by using bypasses of size 2 only. Moreover, on each of the new paths, there will be at most one bypass.

Proof. By Claim 1, every vertex used by two paths in a path matching belongs to some tree. We will show how to resolve a tree of Π by using bypasses of size 2 in such a way that only vertices of the tree are affected. Then we are done by solving all trees independently.

Let Π_T be a subset of Π, forming a tree. For simplicity, we address Π_T itself as a tree. Every vertex of Π_T being a conflict has at least three edges incident to it, since it cannot be endpoint of two paths in Π, and it is part of at least two edge-disjoint paths by definition of a conflict.

We reduce the size of the problem at hand in that we eliminate paths from the tree by resolving conflicts.

Procedure 1
Input: A minimal path matching Π for some vertex set U on (V, E)
fulfilling properties (1) and (2).
For all trees Π_T of Π
While there are conflicts in Π_T (i.e. there is more than one path in Π_T)
pick an arbitrary path $p \in \Pi_T$;
if p has only one conflict v, and v is an endpoint of p,
pick another path using v as new p instead;
let $v_1, v_2, \ldots, v_k$ be (in this order) the conflicts in p;
while $k > 1$
consider two paths $p_1, p_k \in \Pi_T$ which use v_1 respectively v_k, commonly with p;
pick as new p one of p_1, p_k which was formerly not picked;
let $v_1, v_2, \ldots, v_k$ be (in this order) the conflicts in p;
let v be the only vertex of the finally chosen path p which is a conflict;
if v has two incident edges in p,
replace those with a bypass,
else (v is an endpoint of p)
replace the single edge incident to v in p together
with one of the previously picked paths with a bypass (see Figure 4).
Output: the modified conflict-free path matching Π'.

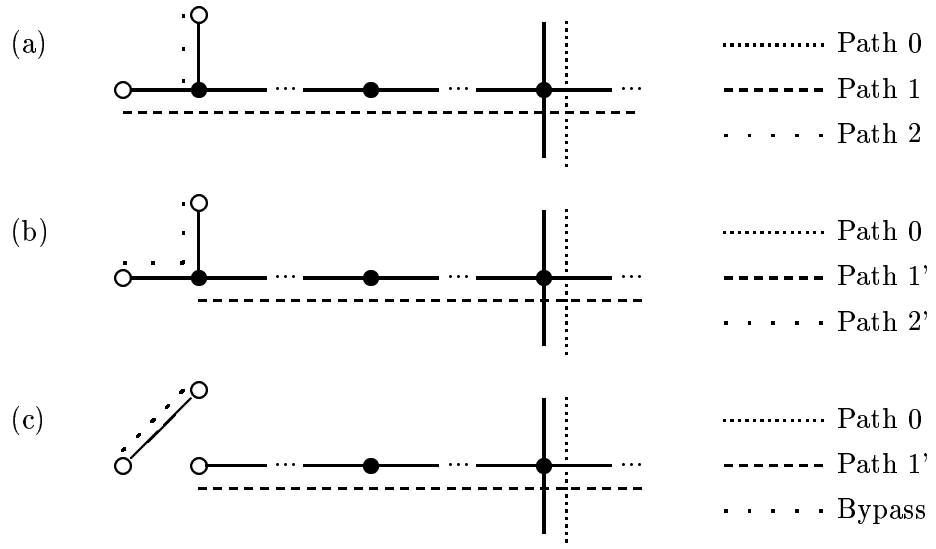

Figure 4: Eliminating a path from a tree. Vertices being conflicts are filled, i.e. non-filled vertices belong to only one path.

The second case of the last step is shown in Figure 4(a) (the paths are visited in the order 0,1,2). The replacement can be considered as first switching some parts of the paths (Figure 4(b)) and then building the bypass as in the "if"-case (Figure 4(c)).

It remains to convince ourselves that the above procedure results in eliminating successively all conflicts within Π by using bypasses of size 2 only.

Since Π_T is a tree, only one of the paths p_1, p_k sharing a vertex with the actual path p may be visited before. Thus the inner while-loop will always end up with a path having only one conflict. Since a new path is chosen w.r.t. the extremal conflict vertices of the previous one, the second case of the case distinction will always be as shown in Figure 4. Note that we use here the property of the path matching that each vertex is endpoint of at most one path of that matching, which guarantees the existence of the edge "borrowed" from Path 1. The existence of Path 1 in turn is guaranteed by the fact that Procedure 1 changes its pick immediately if it has initially picked a path like Path 2.

Since each construction of a bypass reduces the number of paths having conflicts by one, it is clear that the whole procedure will terminate.

The bypass size will always be 2 since each bypass is built between vertices belonging only to one path, and that path is conflict-free afterwards. Consequently, the new edge will never be involved again in subsequent bypass constructions, and every path will finally have only one bypass. □

4.3 Dividing the spanning tree into a forest of degree at most 3

In this section, we describe the implementation of Step 5 of Algorithm PMCA. It divides the minimal spanning tree by resolving conflicts into several trees, whose crucial property is that they have vertices of degree at most 3.

Procedure 2 below is based on the following idea. First, a root of T is picked. Then, we consider a path p_i in T which, under the orientation w.r.t. this root, will go up and down. The two edges immediately before and after the turning point are bypassed. One possible view of this procedure is that the minimal spanning tree is divided into several trees, since each bypass building divides a tree into two.

Procedure 2
Input: T and the paths $p_1, p_3, p_5, \ldots$ computed in Step 4 of Algorithm PMCA.
Choose a node r as a root in T.
For each path
$p_i = (v_1, v_2), (v_2, v_3), ..., (v_{n_i-1}, v_{n_i})$ in T do
  Let v_j be the node in p_i of minimal distance to r in T.
  If $1 < j < n_i$ then
    bypass the node v_j and call this new path p_i'.
  else $p_i' = p_i$.
Output: The paths $p_1', p_3', p_5', \ldots$, building a forest T_f.

Claim 3
If a node v occurs in two different paths p_i' and p_j' of T_f, then v is an inner node in one path and a terminal node in the other path. I.e. the node degree of the forest spanned by $p_1', p_3', p_5', \ldots$ is at most three.

Proof. Let v be a node occurring in paths p'_i and p'_j. The node v cannot be a terminal node in both paths: In T the two paths p_i, p_j cannot have a common endpoint due to Remark 1. And building a bypass in Procedure 2 never changes the endpoints of a path.

Assume that v is an inner node in both paths. Then there exist at least four incident edges $(v, v_1), (v, v_2), (v, v_3), (v, v_4)$ in T. Furthermore at most one of the nodes v_1, v_2, v_3, v_4 is closer to r than v. Thus at least one of the the paths will be bypassed, because v is closer to the root r than all other nodes of that path. This is a contradiction to our assumption. □

From Procedure 2, we obtain immediately the next observation which will be used in the next section.

Remark 2
In T_f, every path has at most one bypass, and every bypass is of size two. □

4.4 Conflict resolution in the Eulerian tour

Remember that after Step 5, we have an Eulerian cycle π' as a sequence of paths $p'_1, p_2, p'_3, p_4, \ldots$ such that the odd numbered paths $p'_1, p'_3, \ldots$ are in T_f, resulting from the spanning tree, and $p_2, p_4, \ldots$ are elements of the path matching Π' (as modified in Step 3).

Before we present Procedure 3, we have a close view on the structure of the paths $p'_1, p'_3, p'_5, \ldots$ and $p_2, p_4, p_6, \ldots$ and state some key observations for Procedure 3.

Claim 4
Vertices which are leaves in T_f are not conflicts in π'.

Proof. A leaf v in T_f occurs by definition only on one path p' of T_f. Being endpoint of p' implies that it is also an endpoint of a path p from Π'. That means p is the continuation of p' in π' and vice versa, so v being endpoint in the two paths means not being a conflict. But also the last possibility of v being a conflict, the existence of another path $\hat{p} \in T_f$ containing v, is excluded since this would be a conflict between p and $\hat{p}$ in T_f which is conflict-free by Claim 2. □

The following claim states the crucial observation which assures that Procedure 3 constructs bypasses in the worst case out of an old bypass of size 2 and two edges not being bypasses before. Thus, we will have bypasses of size at most 4 after all.

Claim 5
In the cycle $p'_1, p_2, p'_3, p_4, p'_5, p_6, \ldots$, between each two bypasses there is at least one vertex not being a conflict.

Note that "between" includes the case that the claimed vertex may be endpoint of one or both edges being bypasses.

Proof. Claim 5 relies itself on another observation about T_f and the output of Procedure 2.

For every path p'_i being output of Procedure 2 holds:

$$\text{If } p'_i \text{ contains a bypass, both of its endpoints are leaves in } T_f, \text{ and} \tag{3}$$

$$\text{otherwise at least one of its end vertices is a leaf.} \tag{4}$$

In view of Claim 4 (leaves in T_f are not conflicts), this implies Claim 5: Each of the paths $p'_1, p_2, p'_3, p_4, p'_5, p_6, \ldots$, alternatingly from Π' and T_f, contains by Claim 2 and Remark 2 at most one bypass. Every other path is from T_f. If it contains a bypass, it provides by (3) at both ends the claimed conflict-free vertices. If it does not contain a bypass, one of its endpoints is by (4) the conflict-free vertex between the two potential bypasses of its neighboring paths from Π'.

Now to show (3) and (4), we just look at an endpoint v of path p'_i. Let v' be its neighbor in that path, i.e. (v, v') is the first edge of p'_i, or (v', v) is the last. If v' is closer to the root r of T then v will be a leaf in T_f for the following reason:

Procedure 2 changes the degree of v by an even number, i.e. v has odd degree in T. If that degree is 1, v is already a leaf in T, and hence in T_f. If the degree of v in T is larger, for all paths in T containing v, other than p'_i, v will be an inner vertex and at the same time the one closest to r ((v, v') is the unique edge from v towards r). Thus, v will by bypassed by Procedure 2 in all paths other than p'_i, becoming a leaf in T_f.

Next, the endpoint v of a path in a tree can only be either the unique vertex closest to the root r, or else it has to be farther away from r than its neighbor v', and hence is a leaf. Only one of the two endpoints can be the unique closest vertex, which shows (4). And a bypass is constructed in a path by Procedure 2, only if neither of the endpoints is the vertex closest to the root, which proves (3). □

Below, we present Procedure 3 which consecutively resolves the remaining conflicts. Note that s, t, u, v, and their primed versions, denote *occurrences* of vertices on a path, rather than the vertices itself. In one step, Procedure 3 has to make a choice. We state the rule used for choosing separately for further reference in proving the correctness of the procedure. In that proof, we will show also that this rule always gives a unique choice.

Remember that the input of Procedure 3 is a cycle $p'_1, p_2, p'_3, p_4, p'_5, p_6, \ldots$, where Claim 5 holds.

Procedure 3

Input: a cycle π' on (V, E) where every vertex of V occurs once or twice.
Take an arbitrary conflict, i.e. a vertex occurring twice as u and u' in π';
bypass one occurrence, say u (with a bypass of size 2);
while there are conflicts remaining
  if occurrence u has at least one unresolved conflict as neighbor
    let v be one of them, chosen by rule (5);
    (i.e. (v, u) or (u, v) is an edge of π' and there is another occurrence v' of

the same vertex as v)
resolve that conflict by bypassing v'
else
resolve an arbitrary conflict;
let u be the bypassed vertex.
Output: the modified cycle π''.

The rule used for choosing the next conflict to be resolved is the following.

> If between u and another bypassed vertex occurrence t on π', there are only unresolved conflicts, chose v to be the neighbor of u towards t. (5)

That there cannot be vertex occurrences satisfying the conditions for t on both sides of u will be shown as part of proving the correctness of Procedure 3.

Claim 6
Procedure 3 terminates after resolving all conflicts, and it generates bypasses of size at most four overall, i.e. taking into account that some edges of the input cycle π' may be bypasses of size 2 themselves.

Proof. Clearly, Procedure 3 will terminate after resolving all conflicts if rule (5) always produces a unique choice when applied. This in turn follows from statement (6) which we prove by induction on the number of conflicts resolved by Procedure 3.

> Assume during the run of Procedure 3, there are two vertex occurrences s, t bypassed by the procedure such that between them, there are only unresolved conflicts.
> Then one of s, t, say s, is the vertex occurrence u just under consideration by Procedure 3, and the vertex occurrence v, being the neighbor of s towards t, is resolved next as not bypassed. (6)

After the first conflict resolution, (6) holds since there is only one vertex occurrence u bypassed, and between u and itself on the cycle π', there will always be u', a resolved, non-bypassed vertex occurrence.

In the induction step, we have to look at a conflict resolution performed by Procedure 3. Here, the applicability of rule (5) relies on the induction hypothesis. Assume there would be to both sides of u vertex occurrences s, t such that between u and both of them there would be only unresolved conflicts. Then there (6) would be applicable to s, t at an earlier stage (before bypassing u there were only unresolved conflicts between s and t), and consequently one of the conflicts in between would have been resolved as *not* bypassed by the induction hypothesis, a contradiction.

Now that rule (5) can be applied, this application just guarantees that (6) still holds after the actual conflict resolution.

Next, we will see how (6) implies that Procedure 3 bypasses at most 2 consecutive vertex occurrences, i.e. never replaces more than three consecutive edges of π'. Again, we show this by contradiction.

We look at any piece $(u_0, u_1), (u_1, u_2), (u_2, u_3), (u_3, u_4)$ of the input cycle π' and assume the three consecutive vertex occurrences u_1, u_2, u_3 are bypassed during the run of Procedure 3.

We consider the situation where Procedure 3 just has bypassed u_2. If any of its neighbors u_1, u_3 would be unresolved at that time, the procedure would choose one of them as the next conflict to be resolved. Then it would resolve that conflict by fixing as *not* bypassed u_1 respectively u_3. Consequently, u_1 and u_3 have to be treated before u_2. W.l.o.g., let u_1 be treated before u_3.

Next, we look at the situation where Procedure 3 just has bypassed u_3. Since u_1 is bypassed earlier, there are only unresolved conflicts (namely u_2) between the bypassed vertices u_1, u_3. By (6), u_2 would be resolved next as *not* bypassed, giving the desired contradiction.

We end this proof by showing how bypassing at most two consecutive vertices limits the overall bypass size.

According to Claim 5, each new bypass may substitute at most one old bypass of size two and two edges being no bypasses before. (Of course, only vertices being conflicts will be bypassed). Overall this limits the bypass size by 4. □

4.5 The performance of the algorithm PMCA

To show the claimed approximation ratio, we start by bounding the cost of the initial objects, constructed in Step 1 and 2 of Algorithm PMCA.

Claim 7 (a) The cost of a minimum spanning tree on $G = (V, E)$ is at most the minimal cost of a Hamiltonian tour.

(b) The cost of a minimum path matching Π for U on $G = (V, E)$ is at most half of the minimal cost of a Hamiltonian tour;

Proof.

(a) Removing one edge from a Hamiltonian tour of minimal cost gives a spanning tree whose cost is at least that of the minimal spanning tree. (Note that the reasoning here is just as in case of the original Christofides algorithm since it is independent of the triangle inequality.)

(b) We take any Hamiltonian tour of minimal cost on G. Cutting it into paths at the vertices of U, we put the paths alternatingly into two sets. Each of these two sets is a path matching for U on G. The cheaper one has at most half of the cost of the Hamiltonian tour but at least the cost of Π. □

Now, we can state the approximation ratio of PMCA.

Claim 8
The cost of the Hamiltonian tour π'' returned by the algorithm PMCA is at most $\frac{3}{2}\beta^2$ times the cost of an optimal one.

Proof. Let C_{opt} be the minimal cost of a Hamiltonian tour in the given graph G. We bound the size of the objects computed by PMCA step by step in comparison to C_{opt}.

In the first two steps, the algorithm computes a spanning tree T and a path matching Π. According to Claim 7, their respective costs are bounded by $cost(T) \leqslant C_{opt}$ and $cost(\Pi) \leqslant 0.5 \cdot C_{opt}$.

In Step 3, Π is replaced with Π' which increases the cost of the path matching by at most a factor of β. As noted at the end of Section 4.2, this is a consequence of using bypasses of size at most two: bypassing means replacing two edges by one, and the cost of the new edge is β times the sum of the costs of the original edges (in a Δ_β-TSP instance).

The Eulerian tour π constructed in Step 4 consists exactly of the edges of T and Π'. Thus its cost is $cost(\pi) = cost(T) + cost(\Pi') \leqslant (1 + 0.5 \cdot \beta) \cdot C_{opt}$.

In Step 5, some edges from T are bypassed by using bypasses of size 2 only. Consequently, the cost of the resulting tour π' can be bounded by $cost(\pi') = cost(T_f) + cost(\Pi') \leqslant (1 + 0.5) \cdot \beta \cdot C_{opt}$.

Finally, in Step 6 sub-paths of π' are replaced by single edges. This is done in a way such that at most 4 of the original edges from T and Π are replaced with a new single one (taking into account the combined effects of Steps 3, 5, and 6). This may (in the given Δ_β-TSP instance) increase the costs by a factor of at most β^2, compared to the sum of the costs of T and Π. Consequently, we have for the resulting Hamiltonian cycle π'' the cost $cost(\pi'') \leqslant \frac{3}{2}\beta^2 \cdot C_{opt}$. □

This concludes the proof of Theorem 1. □

5 Conclusion and Discussion

In the previous sections we have introduced the concept of stability of approximations and we have applied it for TSP. Here we discuss the potential applicability and usefulness of this concept. Applying it, 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).

2. We are not able to successfully apply a given approximation algorithm A (a PTAS) for additional inputs, but one can 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 (see Theorem 1 and [AB95, BC99]).

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 approximation algorithm for $Q_{r,d} = (\Sigma_I, \Sigma_O, L, Ball_{r,d}(L_I), \mathcal{M}, cost, goal)$ under the assumption $P \neq NP$, then this means that the problem Q is "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, which is the main aim of our concept. This has been achieved for TSP now. Collecting results of Theorem 1 and of [BC99], we have $\min\{\frac{3}{2}\beta^2, 4\beta\}$-approximation algorithms for $\Delta_{\beta,dist}$-TSP, and following [BC99], $\Delta_{\beta,dist}$-TSP is not approximable within a factor $1 + \varepsilon \cdot \beta$ for some $\varepsilon < 1$. While TSP does not seem to be tractable from the previous point of view of approximation algorithms, using the concept of approximation stability, it may look tractable for many specific applications.

References

[Ar96] S. Arora: Polynomial time approximation schemes for Euclidean TSP and other geometric problems. In: *Proc. 37th IEEE FOCS*, 1996, pp. 2–11.

[Ar97] S. Arora: Nearly linear time approximation schemes for Euclidean TSP and other geometric problems. In: *Proc. 38th IEEE FOCS*, 1997, pp. 554–563.

[AB95] T. Andreae, H.-J. Bandelt: Performance guarantees for approximation algorithms depending on parametrized triangle inequalities. *SIAM J. Discr. Math.* 8 (1995), pp. 1–16.

[BC93] D. P. Bovet, P. Crescenzi: *Introduction to the Theory of Complexity*, Prentice-Hall 1993.

[BC99] M. A. Bender, C. Chekuri: Performance guarantees for the TSP with a parameterized triangle inequality. In: *Proc. WADS'99, LNCS*, to appear.

[Chr76] N. Christofides: Worst-case analysis of a new heuristic for the traveling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, 1976.

[Co71] S. A. Cook: The complexity of theorem proving procedures. In: *Proc. 3rd ACM STOC*, ACM 1971, pp. 151–158.

[CK98] P. Crescenzi, V. Kann: *A compendium of NP optimization problems.* `http://www.nada.kth.se/theory/compendium/`

[CLR90] T. H. Cormen, C. E. Leiserson, R. L. Rivest: *Introduction to algorithms.* MIT Press, 1990.

[En99] L. Engebretsen: An explicit lower bound for TSP with distances one and two. Extended abstract in: *Proc. STACS'99, LNCS 1563*, Springer 1999, pp. 373–382. Full version in: *Electronic Colloquium on Computational Complexity*, Revision 1 of Report No. 46 (1999).

[GJ79] M. R. Garey, D. S. Johnson: *Computers and Intractability. A Guide to the Theory on NP-Completeness.* W. H. Freeman and Company, 1979.

[Hå97] J. Håstad: Some optimal inapproximability results. In: *Proc. 29th ACM STOC*, ACM 1997, pp. 1–10.

[Ho96] D. S. Hochbaum (Ed.): *Approximation Algorithms for NP-hard Problems.* PWS Publishing Company 1996.

[Hr98] J. Hromkovič: Stability of approximation algorithms and the knapsack problem. Unpublished manuscript, RWTH Aachen, 1998.

[IK75] 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.

[Jo74] D. S. Johnson: Approximation algorithms for combinatorial problems. *JCSS* 9 (1974), pp. 256–278.

[Lo75] L. Lovász: On the ratio of the optimal integral and functional covers. *Discrete Mathematics* 13 (1975), pp. 383–390.

[LLRS85] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, D. B. Shmoys (Eds.): *The Traveling Salesman Problem.* John Wiley & Sons, 1985.

[Mi96] 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.

[MPS98] E. W. Mayr, H. J. Prömel, A. Steger (Eds.): *Lectures on Proof Verification and Approximation Algorithms. LNCS* 1967, Springer 1998.

[Pa77] Ch. Papadimitriou: The Euclidean traveling salesman problem is NP-complete. *Theoretical Computer Science* 4 (1977), pp. 237–244.

[Pa94] Ch. Papadimitriou: *Computational Complexity*, Addison-Wesley 1994.

[PY93] Ch. Papadimitriou, M. Yannakakis: The traveling salesman problem with distances one and two. *Mathematics of Operations Research* 18 (1993), 1–11.

ECCC ISSN 1433-8092
http://www.eccc.uni-trier.de/eccc
ftp://ftp.eccc.uni-trier.de/pub/eccc
ftpmail@ftp.eccc.uni-trier.de, subject 'help eccc'