Algorithmic Operations Research

On the Stability of Approximation for Hamiltonian Path Problems

Luca Forlizzi, Juraj Hromkovi, Guido Proietti et Sebastian Seibert

Volume 1, numéro 1, summer 2006

URI : https://id.erudit.org/iderudit/aor1_1art03

Aller au sommaire du numéro

Éditeur(s)

Preeminent Academic Facets Inc.

ISSN

1718-3235 (numérique)

Découvrir la revue

Citer cet article

Forlizzi, L., Hromkovi, J., Proietti, G. & Seibert, S. (2006). On the Stability of Approximation for Hamiltonian Path Problems. *Algorithmic Operations Research*, *1*(1), 31–45.

Résumé de l'article

We consider the problem of finding a cheapest Hamiltonian path of a complete graph satisfying a relaxed triangle inequality, i.e., such that for some parameter > 1, the edge costs satisfy the inequality c({x, y}) ⩽ `c({x, z}) + c({z, y})´ for every triple of vertices x, y, z. There are three variants of this problem, depending on the number of prespecified endpoints: zero, one, or two. For metric graphs there exist approximation algorithms, with approximation ratio 32 for the first two variants and 53 for the latter one.

Using results on the approximability of the Travelling Salesman Problem with input graphs satisfying the relaxed triangle inequality, we obtain for our problem approximation algorithms with ratio min(2 + , 32 2) for zero or one prespecified endpoints, and 53 2 for two endpoints.

Algorithmic Operations Research Vol.1 (2006) 31–45

On the Stability of Approximation for Hamiltonian Path Problems

Luca Forlizzi, [a] Juraj Hromkovič, [b] Guido Proietti [a,c] and Sebastian Seibert [b]

[a]Dipartimento di Informatica, Università di L'Aquila, I-67010 L'Aquila, Italy
[b]Department Informatik, ETH Zentrum, CH-8092, Zürich, Switzerland
[c]Istituto di Analisi dei Sistemi ed Informatica "A. Ruberti", CNR, Roma, Italy

Abstract

We consider the problem of finding a cheapest Hamiltonian path of a complete graph satisfying a relaxed triangle inequality, i.e., such that for some parameter $\beta > 1$, the edge costs satisfy the inequality $c(\{x, y\}) \le \beta\big(c(\{x, z\}) + c(\{z, y\})\big)$ for every triple of vertices x, y, z. There are three variants of this problem, depending on the number of prespecified endpoints: zero, one, or two. For metric graphs there exist approximation algorithms, with approximation ratio $\frac{3}{2}$ for the first two variants and $\frac{5}{3}$ for the latter one.

Using results on the approximability of the Travelling Salesman Problem with input graphs satisfying the relaxed triangle inequality, we obtain for our problem approximation algorithms with ratio $\min(\beta^2 + \beta, \frac{3}{2}\beta^2)$ for zero or one prespecified endpoints, and $\frac{5}{3}\beta^2$ for two endpoints.

1. Introduction

It often happens that the hardness of the polynomial-time approximability of a problem varies according to the input instance, and some hard problem becomes relatively easy for certain subclasses of instances. Given a hard optimization problem, and a polynomial-time approximation algorithm for a subclass of input instances, a natural idea is trying to extend the approximation algorithm to a wider class of problem instances. This idea is captured by the notion of *stability of approximation*, which provides a formal framework to study the change of the approximation ratio according to a small change in the specification (some parameter, characteristics) of the set of problem instances considered [4].

One of the most successful application of the concept of stability of approximation concerns the famous Travelling Salesman Problem (TSP). It is well known that TSP is not only NP-hard, but also not approximable in polynomial time with constant approximation ratio. But if one considers Δ-TSP, namely TSP for complete input graphs satisfying the triangle inequality (i.e., *metric graphs*), one can design a polynomial time $\frac{3}{2}$-approximation algorithm [5]. To extend the class of input graphs for which the TSP is approximable (in polynomial time, with constant approximation ratio), one considers the so called β-*triangle inequality*. For a given $\beta \ge 1$, a graph (V, E) satisfies the β-triangle inequality if for all vertices u, v, x it is $c(\{u, v\}) \le \beta\big(c(\{u, x\}) + c(\{x, v\})\big)$, where $c : E \mapsto \mathbb{R}^+$ is the cost function of the graph. For every real $\beta > 1$, Δ_β-TSP is the restriction of the TSP to inputs satisfying the β-triangle inequality.

In the past, several polynomial time approximation algorithms providing constant approximation ratio for Δ_β-TSP were proposed. Currently, there are three different algorithms which achieve the smallest approximation ratio, each for a distinct range of values of β:

(A) the REFINED T^3 algorithm, providing a $(\beta^2 + \beta)$ approximation ratio [1], which is the best for $2 \le \beta \le 3$;
(B) the Bender and Chekuri (4β)-approximation algorithm [3], best for $\beta > 3$;
(C) the Path Matching Christofides Algorithm (PMCA) providing a $\frac{3}{2}\beta^2$ approximation ratio [4], best for $1 < \beta < 2$.

In this paper, we study how these results can help to design approximation algorithms for the *Hamiltonian Path Problem* (HPP), where one is required to compute a minimum cost Hamiltonian path spanning a complete graph G. There are three natural variants of the HPP, differing in the constraints imposed to the endpoints of the desired path: they can be both arbitrary vertices (HPP$_0$), or one of them can be constrained to be a pre-

Email: Luca Forlizzi, [forlizzi@di.univaq.it], Juraj Hromkovič, [jh@cs.rwth-aachen.de], Guido Proietti [proietti@di.univaq.it], Sebastian Seibert [seibert@cs.rwth-aachen.de].

specified vertex s (HPP$_1$), or both of them can be constrained to be prespecified vertices s and t (HPP$_2$). The TSP is easily reducible to any of these variants, so they are NP-hard too. For $k \in \{0, 1, 2\}$, we denote the restrictions of HPP$_k$ to input graphs satisfying the triangle and the β-triangle inequalities, respectively by Δ-HPP$_k$ and Δ_β-HPP$_k$. In [8], Hoogeveen applied to the HPP the approach introduced by Christofides for Δ-TSP [5], providing $\frac{3}{2}$-approximation algorithms for Δ-HPP$_0$ and Δ-HPP$_1$, and a $\frac{5}{3}$-approximation algorithm for Δ-HPP$_2$.

In this paper, trying to extend the class of graphs for which HPP is approximable, we consider again the β-triangle inequality and investigate whether each of the three approaches for Δ_β-TSP is suitable also for HPP. To this aim, we concentrate on adapting the approaches of (A) and (C), which distinguish themselves, respectively, by running times $O(n^2)$ and $O(n^3)$, where n is the number of vertices in G. This is acceptable for practical purposes rather than the $O(n^5)$ running time of (B). We just note why the approach of (B) would need some additional considerations in order to be carried over to HPP. The algorithm of Bender and Chekuri is based on results by Fleischner [6,7], who proved that the square of a 2-vertex-connected graph is Hamiltonian, and by Lau [9,10], who provided an effective procedure for the construction of a Hamiltonian cycle. So, Bender and Chekuri first construct an approximation of the minimum cost 2-vertex-connected subgraph, and then apply, on the resulting graph, Lau's procedure to obtain a Hamiltonian cycle. The length of a minimum cost 2-vertex-connected subgraph is a lower bound on the cost of any Hamiltonian cycle, and from this fact the bound on the cost of their solution follows. However, the length of a minimum cost 2-vertex-connected subgraph is not a lower bound on the cost of a Hamiltonian path. Hence, this approach does not lead immediately to an approximation algorithm for the HPP.

The approaches leading to algorithms (A) and (C) are studied in Sections 2. and 3., respectively. For Δ_β-HPP$_0$ and Δ_β-HPP$_1$, we keep with both approaches the same ratio bounds as for the TSP, thus obtaining $\min(\beta^2 + \beta, \frac{3}{2}\beta^2)$-approximation algorithms. For Δ_β-HPP$_2$, using the approach of (C), we achieve a $\frac{5}{3}\beta^2$ approximation ratio which is a natural generalization of the $\frac{5}{3}$ approximation ratio known for metric graphs. With the approach of (A), instead, we obtain approximation ratios worse than $\frac{5}{3}\beta^2$, for any $\beta > 1$. Nevertheless, such an approach is still somehow useful for HPP$_2$, since it allows to obtain an $O(n^2)$ time 3-approximation algorithm for Δ-HPP$_2$, faster than the $O(n^3)$ time previously known approximation algorithm.

Following [8], we let P^* denote an optimal Hamiltonian path without prescribed endpoints, P^*_s denote an optimal Hamiltonian path with a single prescribed endpoint s, and P^*_{st} denote an optimal Hamiltonian path with prescribed endpoints s and t. We denote by $V(G)$ and $E(G)$, respectively, the set of vertices and the set of edges of a graph G. Given a graph G and a collection Π of paths on the vertices of G, we denote by $G \cup \Pi$ the multigraph obtained by adding to G all the edges of each path in Π. We denote by $\mathrm{EndP}(\Pi)$ the set formed by the endpoints of all the paths contained in Π. Given a graph G and edges e and f connecting vertices of G, we denote by $G - e$ and $G + f$ the graphs obtained, respectively, by removing e from G and by inserting f in G. An edge $e \in E(G)$ is *locally minimal* if there is a vertex $v \in V(G)$ such that e is an edge incident on v of minimum cost. We call an occurrence of a vertex in a path γ *internal*, if it is not an endpoint of γ. Given a path γ, we say that a subpath γ' of γ is a *terminal subpath* if one of the endpoints of γ' is also an endpoint of γ. A path in a graph is *elementary* if it does not contain the same vertex more than once.

2. The REFINED T^3 Algorithm for Hamiltonian Path

In 1960, Sekanina proved that for every tree $T = (V, E)$ the graph

$$T^3 = \big(V, \big\{\{x, y\} \mid x, y \in V, \text{ and there exists in } T \text{ a path from } x \text{ to } y \text{ of length at most } 3\big\}\big)$$

contains a Hamiltonian cycle. Let T be a tree and $H = (u_1, u_2, \ldots, u_n)$ be a Hamiltonian cycle contained in T^3. Let $E(H)$ be the set of edges forming H. For any edge $e = \{x, y\}$ of $E(H)$, let p_e be the unique elementary path in T having x and y as endpoints, and let $A_H = \{p_e \mid e \in E(H)\}$. Clearly, each $p_e \in A_H$ has length at most 3.

Starting from Sekanina's result, Andreae and Bandelt designed in [2] a $(\frac{3}{2}\beta^2 + \frac{1}{2}\beta)$-approximation algorithm for Δ_β-TSP. Given a complete graph G and a minimum spanning tree T of G, they were able to construct a Hamiltonian cycle H of T^3, in such a way that each edge of T occurs in exactly two of the paths of A_H, and that it is the middle edge of at most one path of A_H having length 3. Such properties imply that expensive edges of T do not occur in H more often than cheap edges. Then the cost of H is bounded by a factor times the cost of T, which is, in turn, a lower bound for the cost of an optimal Hamiltonian cycle of G. Note that the strategy used by Andreae and Bandelt can be seen as an enhancement, allowing to deal with β-triangle inequality, of the well known Double-Tree 2-approximation algorithm for Δ-TSP, which computes a Hamiltonian cycle from an Eulerian cycle of the multigraph obtained by doubling each edge of a minimum spanning tree of the input graph.

The result of Andreae and Bandelt has been recently improved by Andreae [1], that presented a $(\beta^2 + \beta)$-approximation algorithm for Δ_β-TSP (see Algorithm 1). The main part of such algorithm is Procedure HCT^3(REFINED) (see Algorithm 2.), which we use later.

Algorithm 1 : REFINED T^3

```
Input: A complete graph $G$.
  Find a minimum spanning tree $T$ of $G$.
  Find a Hamiltonian cycle $H$ of $T^3$ calling
    HCT$^3$(REFINED) with inputs $T$ and an arbitrary locally minimal edge of $T$.
Output: A Hamiltonian cycle $H$ of $G$.
```

Algorithm 2 Procedure HCT^3(REFINED)

```
Input: A tree $T$ with $|V(T)| \geq 3$ and a locally minimal edge $e^* = \{a_1, a_2\}$ of $T$.
  Let $T_i$ be the component of $T - e^*$ containing $a_i$ $(i = 1, 2)$.
  For $i = 1, 2$ do
    If $|V(T_i)| \geq 2$, then pick $a'_i \in V(T_i)$ so that $e^*_i = \{a_i, a'_i\}$ is a locally minimal edge of $T_i$
    else if $|V(T_i)| = 1$ then let $a'_i = a_i$.
    If $|V(T_i)| \geq 3$, then
      Recursively call HCT$^3$(REFINED) with inputs $T_i$ and $e^*_i$ obtaining a Hamiltonian cycle $H_i$ of $T^3_i$ containing $e^*_i$.
      Let $p_i = H_i - e^*_i$.
    else let $p_i = T_i$.
  Construct $H$ by concatenating $p_1, p_2, e^*$ and the edge $\{a'_1, a'_2\}$.
Output: A Hamiltonian cycle $H$ of $T^3$ containing $e^*$.
```

The core result obtained by Andreae is the following ([1], Theorem 1): for a tree T with $|V(T)| \geq 3$ and a real number $\beta \geq 1$, suppose T^3 satisfies the β-triangle inequality. Then it is

$$c(H) \leq (\beta^2 + \beta)c(T)$$

where H is a Hamiltonian cycle of T^3 obtained by application of HCT^3(REFINED) to T, and this inequality is strict if $\beta > 1$. HCT^3(REFINED) requires $O(n^2)$ time.

The fact that the cost of the constructed graph is bounded using the cost of T, is particularly interesting for our purposes, since the cost of T is a lower bound for the cost of an optimal Hamiltonian path, too. Indeed, using Andreae's result, we can easily derive approximation algorithms for HPP$_0$ and HPP$_1$ called, respectively, T^3 HPP$_0$ and T^3 HPP$_1$: we first execute Algorithm 1, and then remove, from the resulting Hamiltonian cycle, an arbitrary edge (T^3 HPP$_0$) or an edge incident on s (T^3 HPP$_1$). It is immediate to see that T^3 HPP$_0$ and T^3 HPP$_1$ are correct and that both of them have approximation ratio $(\beta^2 + \beta)$ and require $O(n^2)$ time.

To attack HPP$_2$, the previous strategy needs an adaptation: first compute a Hamiltonian cycle H_{st} containing $\{s, t\}$, and then return the path p_{st} obtained by deleting $\{s, t\}$ from H_{st}. However, with this approach we construct a cycle having a cost bounded against the cost of an optimal Hamiltonian cycle containing $\{s, t\}$, while P^*_{st} does *not* contain $\{s, t\}$. This leads to an increase of the approximation ratio, as exploited in the proof of Theorem 1. Using this adapted strategy, we obtain Algorithm 3 to approximate Δ-HPP$_2$.

Algorithm 3 T^3 Metric-HPP$_2$

```
Input: A complete metric graph $G = (V, E)$ and two vertices $s, t \in V$.
  1: Find a minimum spanning tree $T_{st}$ of $G$ containing $\{s, t\}$.
  2: Find a Hamiltonian cycle $H_{st}$ of $T^3_{st}$ calling HCT$^3$ with inputs $T_{st}$ and $\{s, t\}$.
  3: Find a Hamiltonian path $p_{st}$ of $G$ by removing edge $\{s, t\}$ from $H_{st}$.
Output: A Hamiltonian path $p_{st}$ of $G$ having $s$ and $t$ as endpoints.
```

Note that in Algorithm 3, we use Procedure HCT^3 presented in [2] instead of the improved version HCT^3(REFINED) listed in Algorithm 2.. The two procedures are similar and for a metric graph G, given a spanning tree T and an edge e of T, they both compute a Hamiltonian cycle H_e containing e such that $c(H_e) \leq 2c(T)$. There are two advantages in using HCT^3 instead of HCT^3(REFINED): the former procedure does not require the input edge e to be locally minimal, and it is also more efficient, running in $O(n)$ time.

Theorem 1. *Algorithm 3 is a* 3*-approximation algorithm for* Δ*-HPP*$_2$*. The algorithm runs in* $O(n^2)$ *time.*

Proof. Let G be a metric graph. Given a Hamiltonian cycle $\bar{H}$ of G containing $\{s,t\}$, $\bar{H}-\{s,t\}$ is a Hamiltonian path of G having s and t as endpoints, and therefore $c(P^*_{st}) \le c(\bar{H}-\{s,t\})$. It follows that $H^*_{st} = P^*_{st}+\{s,t\}$ is an optimal Hamiltonian cycle containing $\{s,t\}$. Let T_{st} be the minimum spanning tree computed in Step 1 and H_{st} be the Hamiltonian cycle computed in Step 2. As shown in [2], we have $c(p_{st})+c(\{s,t\}) = c(H_{st}) \le 2c(T_{st})$. Since by deleting from H^*_{st} an edge different from $\{s,t\}$ one obtains a tree containing $\{s,t\}$, it is $c(T_{st}) < c(H^*_{st}) = c(P^*_{st}) + c(\{s,t\})$. Then we have $c(p_{st}) < 2c(P^*_{st}) + c(\{s,t\}) \le 3c(P^*_{st})$, where the last inequality follows from the triangle inequality. Since HCT^3 runs in $O(n)$ time, the time complexity of the whole algorithm is dominated by the $O(n^2)$ time required to compute T_{st}. □

Although Algorithm 3 has a poor approximation guarantee, it deserves some interest being more efficient than the $O(n^3)$ time algorithm derived in [8] from Christofides' one. We remark that straightforward adaptations of the Double-Tree algorithm to HPP$_2$, construct a graph with proper vertex degrees by doubling the edges of a minimum spanning tree containing $\{s,t\}$. Therefore, there are similar problems in lowering their approximation ratios as with Algorithm 3. Hence it is not immediate to design a linear time approximation algorithm for Δ-HPP$_2$ with a better approximation ratio.

Unfortunately, Algorithm 3 does not provide an approximation guarantee if the input graph does not satisfy the triangle inequality, because in a general graph the cost of $\{s,t\}$ can not be bounded using $c(P^*_{st})$. To extend Sekanina's approach to Δ_β-HPP$_2$, we need another idea. Suppose we have a Hamiltonian path γ spanning G, with cost bounded by a factor times $c(P^*_{st})$. We can transform it into a Hamiltonian path having s and t as endpoints, still having a cost bounded by a factor times $c(P^*_{st})$. W.l.o.g., let $\gamma = (w,\dots,s,s_1,\dots,t_1,t,\dots,z)$. To obtain a path γ' having s as endpoint, we proceed as follows. Consider the terminal subpath $\gamma_s = (w,\dots,s,s_1)$ of γ, and let G_s be the subgraph of G induced by the vertices occurring in γ_s. Since γ_s is a tree containing $\{s,s_1\}$, the cost of a minimum spanning tree T_s of G_s containing $\{s,s_1\}$ is a lower bound for $c(\gamma_s)$. Using Procedure HCT^3(REFINED), we compute a Hamiltonian cycle H_s of G_s containing $\{s,s_1\}$ such that $c(H_s) \le (\beta^2+\beta)c(T_s) \le (\beta^2+\beta)c(\gamma_s)$. Then, by replacing γ_s with H_s in γ, we obtain a graph where s_1 is the only vertex having degree 3. By removing $\{s,s_1\}$, we have the desired path γ'. The same operations can be repeated for the other prescribed endpoint t, leading to Algorithm 4 and the following theorem.

Algorithm 4 T^3 HPP$_2$

```
Input: A complete graph $G=(V,E)$ and two vertices $s,t \in V$.
1: Compute a Hamiltonian path $\gamma = (w,\dots,s,s_1,\dots,t_1,t,\dots,z)$ of $G$.
2: Let $\gamma_s = (w,\dots,s,s_1)$, $\gamma_t = (t_1,t,\dots,z)$ be terminal subpaths of $\gamma$, denote by $G_s$ and $G_t$ the subgraphs of $G$ induced by the vertices occurring, respectively, in $\gamma_s$ and $\gamma_t$.
3: Compute minimum spanning trees $T_s$ of $G_s$ containing $\{s,s_1\}$ and $T_t$ of $G_t$ containing $\{t_1,t\}$.
4: Compute Hamiltonian cycles $H_s$ of $G_s$ and $H_t$ of $G_t$ containing, respectively, $\{s,s_1\}$ and $\{t_1,t\}$.
5: Let $\pi_s = H_s - \{s,s_1\}$, $\pi_t = H_t - \{t_1,t\}$.
6: Compute $\pi_{st}$ by replacing in $\gamma$ subpaths $\gamma_s$ with $\pi_s$ and $\gamma_t$ with $\pi_t$.
Output: A Hamiltonian path $\pi_{st}$ of $G$ having $s$ and $t$ as endpoints.
```

Theorem 2. *For every* $\beta > 1$*, Algorithm 4 is a* $\big((\beta^2+\beta)\min(\beta^2+\beta,\frac{3}{2}\beta^2)\big)$*-approximation algorithm for* Δ_β*-HPP*$_2$*. The algorithm runs in* $O(n^3)$ *time.*

Proof. Let G be a graph satisfying the β-triangle inequality. A Hamiltonian path γ of G can be computed using Algorithm T^3 HPP$_0$ or the $(\frac{3}{2}\beta^2)$-approximation algorithm which we present in Section 3.. Hence $c(\gamma) \le \min(\beta^2+\beta,\frac{3}{2}\beta^2)c(P^*) \le \min(\beta^2+\beta,\frac{3}{2}\beta^2)c(P^*_{st})$. Since γ_s (resp. γ_t) is a tree containing $\{s,s_1\}$ (resp. $\{t,t_1\}$), we have $c(T_s) \le c(\gamma_s)$ (resp. $c(T_t) \le c(\gamma_t)$). The Hamiltonian cycles H_s and H_t can be computed using Algorithm 2.. Note that π_s (resp. π_t) is a Hamiltonian path spanning the same vertices as γ_s (resp. γ_t) and having s and s_1 (resp. t and t_1) as endpoints. By Andreae's result we have $c(\pi_s) < c(H_s) \le (\beta^2+\beta)c(T_s) \le (\beta^2+\beta)c(\gamma_s)$ and, analogously, $c(\pi_t) < (\beta^2+\beta)c(\gamma_t)$. In the last step, π_{st} is obtained replacing the subpaths γ_s and γ_t of γ with, respectively, π_s and π_t. Hence it is $c(\pi_{st}) < (\beta^2+\beta)c(\gamma) \le (\beta^2+\beta)\min(\beta^2+\beta,\frac{3}{2}\beta^2)c(P^*_{st})$.

In Step 1, Algorithm T^3 HPP$_0$ or the $(\frac{3}{2}\beta^2)$-approximation algorithm which we present in Section 3. are used to compute γ. The former algorithm runs in $O(n^2)$ time, while the latter in $O(n^3)$ time. Algorithm 2. used in Step 4 runs in $O(n^2)$ time, and all remaining steps can be trivially performed in $O(n^2)$ time. Hence the whole algorithm runs in $O(n^3)$ time. □

3. The PMCA for Hamiltonian Path

The PMCA is a $(\frac{3}{2}\beta^2)$-approximation algorithm for Δ_β-TSP, inspired by Christofides' algorithm for Δ-TSP. The rough idea of both Christofides' and PMCA algorithms, is the following: first compute a multigraph H with all vertices of even degree, having a cost bounded by $\frac{3}{2}$ times the cost of an optimal Hamiltonian cycle, then compute an Eulerian cycle of H (it has the same cost), and finally transform the Eulerian cycle in a Hamiltonian one by *resolving all conflicts* in it, i.e., by removing repeated occurrences of vertices in the cycle. The final task is trivial in the case of Christofides' algorithm, but not for the PMCA. Indeed, given the β-triangle inequality, with $\beta > 1$, the bypassing of some vertices in a path may increase the cost of the path.

To illustrate the conflict resolution performed as last task of the PMCA we need some formal definitions. Let $G = (V, E)$ be a complete graph. A *path matching* for a set of vertices $U \subseteq V$ is a collection Π of paths having as endpoints distinct vertices of U. The vertices of U which are not endpoints of some path in Π, are said to be *left exposed by* Π. Assume that $p = (u_0, u_1, u_2, \ldots, u_{k-1}, u_k)$ is a path in G, not necessarily simple. A *bypass* for p is an edge $\{u, v\}$ from E, replacing a subpath $(u_i, u_{i+1}, u_{i+2}, \ldots, u_{j-1}, u_j)$ of p from $u = u_i$ to $u_j = v$ $(0 \leq i < j \leq k)$. Its *size* is the number of replaced edges, i.e. $j - i$. Also, we say that the vertices $u_{i+1}, u_{i+2}, \ldots, u_{j-1}$ *are bypassed*. A vertex which occurs at least twice in a path π, or in a cycle γ, or in a set of paths Π, is said to be a *conflict* (respectively in π, γ or Π). We say that a set of paths is *vertex-disjoint* (resp. *edge-disjoint*) if the paths contained in it are elementary and pairwise vertex-disjoint (resp. edge-disjoint).

The PMCA succeeds in bounding by a factor β^2 the cost increase due to conflict resolution, by ensuring, with non trivial techniques, that at most 4 consecutive edges of the Eulerian cycle are substituted with a new one. In detail, H is the union of a minimum spanning tree T and a path matching Π for the set of all vertices of odd degree in T. The Eulerian cycle π of H can be seen as a sequence of paths $p_1, q_1, p_2, q_2, \ldots$ such that $p_1, p_2, \ldots$ are paths in T and $q_1, q_2, \ldots \in \Pi$. The conflict resolution process is realized in three steps:

((i)) conflicts within Π are resolved obtaining a vertex-disjoint set of paths Π';

((ii)) some of the conflicts within paths in T are resolved so that the cycle π' obtained by modifying π according to steps (i) and (ii), contains at most 2 occurrences of each vertex;

((iii)) all remaining conflicts in π' are resolved, by bypassing at most 2 consecutive vertices.

Combining the ideas of [8] and [4], we obtain an approximation algorithm for the Δ_β-HPP$_k$, with $k \in \{0, 1, 2\}$ (see Algorithm 5).

Algorithm 5 PMCA-HPP$_k$

```
Input: A complete graph $G = (V, E)$ with cost function $c : E \mapsto \mathbb{R}^+$ and a set $A$ of $k$ prespecified endpoints $(0 \leq k \leq 2)$.
1: Construct a minimum spanning tree $T$ of $G$.
2: Let U be the set composed by vertices of $A$ having even degree in $T$ plus vertices of $V \setminus A$ having odd degree in $T$; construct a minimum (edge-disjoint) path matching $\Pi$ for $U$, leaving $2 - k$ vertices of $U$ exposed. If necessary, remove an edge from $T$, so that the multigraph $T \cup \Pi$ has 2 odd degree vertices, which we denote by $w$ and $z$ (observe that any prespecified endpoint is among $w$ and $z$).
3: Resolve conflicts in $\Pi$ (using bypasses of size 2 only), in order to obtain a vertex-disjoint path matching $\Pi'$ such that $z$ can only occur as an endpoint of a path in $\Pi'$.
4: Construct an Eulerian path $\pi$ of $H = T \cup \Pi'$ having $w$ and $z$ as endpoints ($\pi$ can be considered as a sequence of alternating paths from $T$ and $\Pi'$, where $p_1, p_2, \ldots$ are the paths in $T$ and $q_1, q_2, \ldots \in \Pi'$).
5: Resolve conflicts inside the paths $p_1, p_2, \ldots$ obtaining the modified paths $p'_1, p'_2, \ldots$ and the modified Eulerian path $\pi'$, so that $T$ is divided into a forest $T_f$ of trees of degree at most 3, $w$ and $z$ are the endpoints of $\pi'$, and $z$ is not a conflict in $\pi'$ (conflict resolution in this step is done using bypasses of size 2 only).
6: Resolve every remaining conflicts in $\pi'$ using bypasses of overall size 4 (where overall means that a bypass constructed in any previous step counts for 2 edges), obtaining a Hamiltonian path $\pi''$ having $w$ and $z$ as endpoints.
Output: A Hamiltonian path $\pi''$ of $G$ having $w$ and $z$ as endpoints.
```

Similarly to the PMCA, Algorithm 5 computes a multigraph H with all vertices but 2 of even degree. The 2 odd degree vertices include any prespecified endpoint. Since Π' is vertex-disjoint, in H there can be at most 2 edges between a pair of vertices, one from T and one from a path of Π'. In the following description, it will be clear from the context whether edges we refer to are contained in T or in a path of Π'. Algorithm 5 proceeds by constructing an Eulerian path π of H, having the odd degree vertices as endpoints. Finally, conflicts are resolved obtaining a Hamiltonian path.

Here, the conflict resolution process can not be realized as in the PMCA. In particular, in step (iii) of the conflict resolution process in PMCA, for each conflict there is complete freedom in choosing which of the 2 vertex occurrences to bypass. To avoid that more than 2 consecutive vertices of π' are bypassed, PMCA relies exactly on this freedom. In our problem, we loose part of such freedom, since it may happen that the endpoints of π' are conflicts: in this case, we are not allowed to bypass the occurrences which are endpoints of π', hence we are forced to bypass the internal ones. Although the problem regards only two vertices, it may render impossible to resolve all conflicts bypassing at most 2 consecutive vertices, as the following example shows.

In Figure 1, w_1, w_2, (as well as z_1, z_2 and v_1, v_2) denote distinct occurrences in π of the same vertex. Since we are forced to bypass both w_2 and z_1, no matter which one of v_1, v_2 we bypass, there would be 3 consecutive bypassed vertices in the Hamiltonian path, causing the cost to increase more than a factor β^2. To avoid such situations, and resolve all conflicts in π' by bypassing at most 2 consecutive vertices, we have to change the whole conflict resolution process, as described in the following. Step 1 of Algorithm 5 is trivial, while the remaining ones deserve a detailed description.

Step 2 For any $u, v \in V$, let the cost of a cheapest path between u and v be denoted by $d(u, v)$. To construct a path matching Π which leaves k vertices exposed, we first compute all-pairs cheapest paths in G. Then we define a complete graph G' on U augmented with $2-k$ dummy vertices, with a cost function c' specified as follows:

$$c'(u,v) = \begin{cases} \infty & \text{if } u \text{ and } v \text{ are distinct} \\ & \text{ dummy vertices} \\ 0 & \text{if } u \in U \text{ and } v \text{ is dummy} \\ d(u,v) & \text{if } u, v \in U \end{cases}$$

Next, we compute a minimum matching M on G', we remove from it edges incident on dummy vertices, and finally we include in Π, for each edge $\{u, v\}$ of M, the cheapest path in G between u and v. Clearly, this can be done in $O(n^3)$ time and results in a minimum path matching on U that leaves $2-k$ vertices exposed. In [4] it is shown that a minimum path matching Π is edge-disjoint and paths within it form a forest.

Consider the multigraph $T \cup \Pi$. This multigraph is connected and has two or zero odd-degree vertices. The latter case occurs only if: there is a single prespecified endpoint s, s has even degree in T (so it belongs to U), and s is left exposed by Π. In this case we remove an arbitrary edge of T incident on s. Let w and z be the two odd-degree vertices in the obtained multigraph. It can be easily seen that any prespecified endpoint is contained in $\{z, w\}$.

We now introduce some simple definitions and observations often used in the following. Given a vertex $v \in V$ we define the *distance in T of v from z*, as the number of edges in the unique elementary path existing in T from v to z, prior to the possible removal, discussed above, of an edge incident on s from T. We denote by y the unique vertex among the neighbors of w in T before the possible removal from T of an edge incident on s, having distance in T from z less than the distance in T of w from z. In the PMCA-HPP$_k$ many paths existing in T, in Π, or in a set of paths S are modified bypassing some of their vertices (see, for example, Algorithm Decompose-Tree). To shorten the exposition, from now on we say that a path p *has a bypass* (resp. *has k bypasses*) meaning that p has been obtained picking a path from T, Π, or S (it will be clear from the context) and applying to it one bypass (resp. k bypasses).

An important observation, used several times in the following, is that since T is a tree, an occurrence of w which belongs to a path p in T, is not the vertex of p having minimum distance in T from z, if and only if p contains the edge $\{w, y\}$. Also note that in case we have a single prespecified vertex s and, as discussed above, we need to remove from T an edge incident to s to have in $T \cup \Pi$ the 2 odd degree vertices z and w, it is $y = z$ and $\{w, y\}$ is exactly the removed edge: this implies that given a path p contained in T after the removal of $\{w, y\}$, if p contains w, then w is the vertex of p having minimum distance in T from z.

Step 3 To perform Step 3 of the algorithm, i.e., to modify path matching Π into a vertex-disjoint one, we use a strategy different from the one employed in the PMCA. The reason is that we have the additional requirement that at least one of the two odd-degree vertices that exist in $T \cup \Pi$ after Step 2, say z, does not have internal occurrences on paths in Π'. In the rest of the description of Step 3, since we only deal with Π, to shorten the exposition we simply write *conflict* to mean

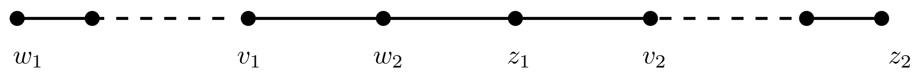

Fig. 1. Impossibility of conflict resolution bypassing at most 2 consecutive vertices

conflict in Π. Given a set of paths $S \subseteq \Pi$ we denote by T_S the graph formed by all edges contained in any of the paths of S. By [4] we know that T_Π is a forest. As in Procedure 1 of [4], we process each connected component of T_Π separately. To this aim, here we use Algorithm Decompose-Tree (see Algorithm 6) which, given an edge-disjoint set of paths, computes a new set of paths with the same set of endpoints, such that on each new path there is at most one bypass of size 2, and on one of the new paths there are no bypasses. More precisely, we prove the following lemma.

Algorithm 6 Decompose-Tree

```
Input: A vertex $x$ and an edge-disjoint set of paths $S = \{q_1, \ldots, q_l\}$ with distinct endpoints such that $T_S$ is a tree and $q_1$ contains $x$.
  Let $S' = \emptyset$ and $q'_1 = q_1$. Let $C$ be the set of vertices of $q_1$ which are conflicts.
  **While** there is at least one vertex in $C$ **do**
    Let $v$ be a vertex in $C$ having maximum distance in $q'_1$ from $x$ (i.e., such that the elementary subpath of $q'_1$ having $x$ and $v$ as endpoints is of maximum cardinality). Extract $v$ from $C$. W.l.o.g. assume $q'_1 = (u_a, \ldots, x, \ldots, v, u_b, \ldots, u_c)$ where $u_b, \ldots, u_c$ are not contained in $C$ in the current iteration (and in the successive ones).
    Let $q_{i_1}, \ldots, q_{i_h}$ be the paths forming the connected component of $T_{S \setminus \{q_1\}}$ such that $q_{i_1}$ contains $v$.
    Call recursively Decompose-Tree with vertex $v$ and set $\{q_{i_1}, \ldots, q_{i_h}\}$ as input, obtaining as result the set of paths $\{q'_{i_1}, \ldots, q'_{i_h}\}$.
    **If** $v$ is internal to $q'_{i_1}$ **then** bypass $v$ from $q'_{i_1}$
    **else** assuming w.l.o.g. $q'_{i_1} = (y, \ldots, y', v)$ modify $q'_{i_1}$ and $q'_1$ as follows: $q'_{i_1} = (y, \ldots, y', u_b, \ldots, u_c)$ and $q'_1 = (u_a, \ldots, x, \ldots, v)$.
    Insert paths $q'_{i_1}, \ldots, q'_{i_h}$ in $S'$.
  Insert $q'_1$ in $S'$.
Output: A set of pairwise vertex-disjoint paths $S' = \{q'_1, \ldots, q'_l\}$ such that paths in $S'$ have the same set of endpoints as those in $S$, and $q'_1$ contains $x$.
```

Lemma 1. *Let S be an edge-disjoint set of paths which have distinct endpoints and form a tree T_S. Let x be a vertex occurring in some of the paths in S. Algorithm Decompose-Tree computes a set S' of pairwise vertex-disjoint paths such that:*

((i)) $\mathrm{EndP}(S) = \mathrm{EndP}(S')$;
((ii)) each path in S' is obtained applying at most one bypass to an elementary path from T_S, and the bypass is of size 2;
((iii)) vertex x occurs on a path in S' obtained picking an elementary path from T_S with no bypasses applied.

Proof. We begin the proof with some easy preliminary considerations. First, observe that at each while-loop iteration, one of the conflicts contained in q_1 is taken into consideration. The set C contains at any time the conflicts occurring in q_1 and not yet taken into consideration. Vertices are inserted in C only before the while-loop begins, and at each iteration a vertex is extracted from C. Hence C eventually becomes empty and the algorithm halts. This also means that any vertex of q_1 which is a conflict in S, is considered in exactly one of the iterations.

For the algorithm to be well defined, at any iteration vertices contained in C have to occur in q'_1. Such a path at the beginning of the algorithm is a copy of q_1 (so any vertices in C occurs in it), and can be later modified only in the else-case of the if-then-else statement, by removing one of its terminal subpaths. The fact that at any iteration vertex v is chosen having maximum distance in q'_1 from x, ensures that vertices removed from q'_1 are, at the moment of the removal, not contained in C (they may have been contained in C during previous iterations). Hence, at any iteration, all vertices contained in C occur on q'_1. This also implies that the vertices in q'_1 are always a subset of those in q_1 and that vertex v taken under consideration at the beginning of any of the iterations, is contained in q'_1.

Since paths in S form a tree T_S, paths in $S \setminus \{q_1\}$ are partitioned in connected components of a forest. For the same reason, there is a bijection between the conflicts occurring in q_1 and the connected components of $T_{S \setminus \{q_1\}}$, relating each of these conflicts with the connected component containing it. At each while-loop iteration, the algorithm considers paths forming the connected component of $T_{S \setminus \{q_1\}}$ corresponding to the conflict v under consideration. Therefore, any path in $S \setminus \{q_1\}$ is considered in exactly one of the while-loop iterations.

In the if-then-else statement, v is removed from q'_{i_1}. Note that in the else-case of such a statement, the changes to q'_1 and q'_{i_1} can be considered as first moving the terminal subpath $(u_b, \ldots, u_c)$ from q'_1 to q'_{i_1} and then bypassing v from q'_{i_1}.

To prove the lemma, we proceed by induction on the cardinality of S. If $|S| = 1$, then $S = \{q_1\}$ contains no conflicts, hence the algorithm halts and returns the input path unmodified. Therefore, in this case the lemma is true.

Otherwise, let us assume that the lemma is true for sets of less than $|S|$ paths. We first show that (i) holds. To this aim, we prove that at the beginning of each while-loop iteration (i.e., before the while-loop condition is evaluated) the invariant $\mathrm{EndP}(S) = \mathrm{EndP}(S') \cup \mathrm{EndP}(\{q'_1\}) \cup_{p \in D} \mathrm{EndP}(p)$ holds, where D is the set of paths forming the connected components of $T_{S \setminus \{q_1\}}$ not yet considered by the algorithm. This is certainly true at the beginning of the first iteration ($S' = \emptyset$ and $q'_1 = q_1$). Assume now the invariant true at the beginning of a generic iteration, where the connected component formed by $q_{i_1}, \ldots, q_{i_h}$ is considered. By the inductive hypothesis, when the recursive call returns, it is $\mathrm{EndP}(\{q_{i_1}, \ldots, q_{i_h}\}) = \mathrm{EndP}(\{q'_{i_1}, \ldots, q'_{i_h}\})$. Paths $q'_{i_2}, \ldots, q'_{i_h}$ are then inserted in S' without modifying their endpoints. In the if-then-else statement of the iteration, it may be (in the else-case) that q'_1 and q'_{i_1} exchange one of their endpoints, but the set $\mathrm{EndP}(\{q'_1, q'_{i_1}\})$ is not modified. Then q'_{i_1} is inserted in S' and no further modified, hence the invariant holds at the beginning of next iteration. Therefore, at the end of the last iteration the invariant holds too. Since at that point $D = \emptyset$, this clearly implies that at the end of the whole algorithm it is $\mathrm{EndP}(S) = \mathrm{EndP}(S')$.

We now prove (iii). At the beginning of the algorithm, x occurs on q'_1, which is, at that time, an exact copy of the elementary path q_1, with no bypasses applied. At each while-loop iteration, path q'_1 can be modified only in the else-case of the if-then-else statement. The modification consists in removing from q'_1 a terminal subpath not containing x. This does not create any bypass on q'_1, and q'_1 remains elementary. Hence at the end of the algorithm x still occurs on q'_1 and (iii) holds.

To prove (ii), we first observe that paths are not modified after their insertion in S'. At each iteration, paths $q'_{i_1}, \ldots, q'_{i_h}$ are inserted in S'. By the inductive hypothesis, at the end of the recursive call, paths $q'_{i_2}, \ldots, q'_{i_h}$ are obtained applying at most one bypass to an elementary path from T_S, and the bypass is of size 2. These paths are inserted in S' with no further modifications. When the recursive call returns, q'_{i_1} does not contain vertices of q_1 other than v, and by the inductive hypothesis, it is obtained picking an elementary path from T_S with no bypasses applied. In the if-then-else statement, subpath $(u_b, \ldots, u_c)$ is possibly appended to q'_{i_1}, which remains elementary since $u_b, \ldots, u_c$ are vertices of q_1 different from v. In the same statement, v is removed from q'_{i_1} with a single bypass of size 2. Successively, q'_{i_1} is inserted in S' and no further modified. Hence any path inserted in S' during the while-loop satisfies conditions prescribed by (ii). The only other path inserted in S' is q'_1. Since, by (iii), it is elementary and no bypasses are ever applied to it, (ii) holds.

Finally, we prove that the algorithm returns a set S' of pairwise vertex-disjoint paths. To this aim, we consider the situation at the beginning of each while-loop iteration and show that the following invariant (formed by the conjunction of two conditions) is always satisfied:

(I1) S' is vertex-disjoint;
(I2) for any path p in S', (I2.1) and (I2.2) hold, where
(I2.1) any vertex u contained in p is not contained in q'_1
(I2.2) any vertex u contained in p is contained in q_1 or in the connected component of $T_{S \setminus \{q_1\}}$ considered in the same while-loop iteration where p was inserted in S' (possibly u is contained in both).

The invariant is certainly true at the beginning of the first iteration ($S' = \emptyset$). We assume that the invariant holds at the beginning of a generic iteration where the algorithm considers a conflict v and the corresponding connected component of $T_{S \setminus \{q_1\}}$, formed by paths $q_{i_1}, \ldots, q_{i_h}$, and we show that at the beginning of the next iteration (i.e., before the while-loop condition is evaluated) the invariant holds still.

We first prove that (I2) is satisfied, namely that each path contained in S' at the beginning of the next iteration satisfies (I2.1) and (I2.2). A path $\bar{q}$ contained in S' in the previous iteration, satisfies (I2.1) and (I2.2) by the invariant, and it is not modified during the current iteration. Since q_1 and the connected components of $T_{S \setminus \{q_1\}}$ are static objects during the algorithm's execution, $\bar{q}$ satisfies (I2.2) also at the end of the current iteration. Path q'_1 can be modified, instead, but this is done only by removing vertices from it, so $\bar{q}$ satisfies also (I2.1) at the end of current iteration.

We now show that (I2.1) and (I2.2) are satisfied also by paths $q'_{i_1}, \ldots, q'_{i_h}$ inserted in S' in the current iteration. Using the inductive hypothesis on $q'_{i_1}, \ldots, q'_{i_h}$

returned by the recursive call, we have that

$$q'_{i_1},\dots,q'_{i_h} \text{ are pairwise vertex-disjoint and } q'_{i_1} \text{ contains } v \quad \text{(A)}$$

and that

$$\text{for } 1 \le k \le h,\ q'_{i_k} \text{ contains a subset of the vertices contained in } q_{i_k}. \quad \text{(B)}$$

By elementary tree properties, we have that

$$\text{if any of } q_{i_1},\dots,q_{i_h} \text{ shares a vertex with } q_1, \text{ then that vertex is } v. \quad \text{(C)}$$

Hence, (A), (B) and (C) together imply that

$$q'_{i_2},\dots,q'_{i_h} \text{ do not share vertices with } q_1. \quad \text{(D)}$$

Paths $q'_{i_2},\dots,q'_{i_h}$ are inserted in S' exactly as they are returned from the recursive call. Then (D) and the fact that vertices in q'_1 are a subset of those in q_1, imply that $q'_{i_2},\dots,q'_{i_h}$ satisfy (I2.1), while (B) implies that they satisfy (I2.2). Path q'_{i_1} is instead modified before the insertion in S', by removing v and possibly appending $(u_b,\dots,u_c)$ to it. Then, since in the else-case $u_b,\dots,u_c$ are at the same time removed from q'_1, from (B), (C) and the fact that vertices in q'_1 are a subset of those in q_1, it follows that q'_{i_1} satisfies (I2.1). From (B) and the fact that all vertices inserted in q'_{i_1} in the else-case are contained in q_1, it follows that q'_{i_1} satisfies (I2.2).

To prove that (I1) holds at the beginning of next iteration, we first observe that (ii) and (iii) imply that any path inserted in S' is elementary. Then we prove that any two paths in S' are vertex disjoint. Two paths contained in S' in a previous iteration are not modified, so they are still vertex-disjoint. Two paths from $q'_{i_1},\dots,q'_{i_h}$ are vertex-disjoint because of (A) and, should $u_b,\dots,u_c$ be inserted in q'_{i_1}, because of (D). It remains to prove that a path $\bar{q}$ inserted in S' in a previous iteration does not share vertices with any of $q'_{i_1},\dots,q'_{i_h}$. Since the invariant was true in previous iterations, $\bar{q}$ satisfies (I2.2). This means, since also $q'_{i_1},\dots,q'_{i_h}$ satisfy (I2.2), that any vertex shared by $\bar{q}$ and one of $q'_{i_1},\dots,q'_{i_h}$ has to be contained in q_1. Then, (D) implies that $\bar{q}$ does not share vertices with any of $q'_{i_2},\dots,q'_{i_h}$. If the then-case occurs, q'_{i_1} does not contain, when inserted in S', vertices of q_1. Hence it does not share vertices with $\bar{q}$, too. If the else-case occurs, q'_{i_1} contains vertices $u_b,\dots,u_c$ of q_1. But in the previous iteration such vertices were contained in q'_1. Hence they are not contained in $\bar{q}$ because in the previous iteration $\bar{q}$ satisfied (I2.1). So we conclude that (I1) holds.

Since the invariant is true after any iteration, it is so after the last one, too. Therefore, at that point, S' is vertex-disjoint and no paths in it share vertices with q'_1. So S' remains vertex-disjoint also in the last step of the algorithm, when q'_1 is inserted into it.

□

Step 3 is realized by applying Algorithm Decompose-Tree to each connected component of the forest T_Π (see Algorithm 7). Property (iii) shown in Lemma 1 is used to ensure that no internal occurrences of z exist on any path in Π'.

Algorithm 7 Procedure Implementing Step 3

```
Input: A minimum path matching $\Pi$ on $G$.
 For any $S \subseteq \Pi$ such that $T_S$ is a connected component of $T_\Pi$
   If $T_S$ contains $z$ then
     Call Decompose-Tree with inputs $z$ and $S$.
     Let $q'_1$ be the unique path containing $z$ in the returned set of paths $S'$.
     If the occurrence of $z$ is internal to $q'_1$, remove $z$ from $q'_1$ with a bypass of size 2.
   else choose an arbitrary vertex $x$ in $T_S$ and call Decompose-Tree with inputs $x$ and $S$.
Output: A conflict-free path matching $\Pi'$ containing no internal occurrences of $z$.
```

Lemma 2. *Let Π' be the set of paths computed as a result of Step 3. Then Π' is vertex-disjoint, $\mathrm{EndP}(\Pi') = \mathrm{EndP}(\Pi)$, and there are no internal occurrences of z on paths in Π'. Moreover, every path in Π' has at most one bypass and every bypass is of size 2.*

Proof. Algorithm 7 calls Decompose-Tree on all $S \subseteq \Pi$ forming a connected component of T_Π. Since $\mathrm{EndP}(\Pi)$ is the union of $\mathrm{EndP}(S)$ for all S forming a connected component of T_Π, the facts that Π' is vertex-disjoint and that $\mathrm{EndP}(\Pi') = \mathrm{EndP}(\Pi)$ follow from Lemma 1. Since T_Π is a forest, there is at most one set $S_z \subseteq \Pi$ forming a connected component of T_Π which contains z. By Lemma 1, the call of Decompose-Tree with inputs z and S_z returns a set of vertex-disjoint paths such that z occurs only on a path q'_1 having no bypasses. If the occurrence of z is internal to q'_1, it is bypassed with a bypass of size 2 and q'_1 will have one bypass. By Lemma 1, all other paths returned by some Decompose-Tree call, have at most one bypass of size 2. □

Step 4 In $H = T \cup \Pi'$, w and z are the only vertices of odd degree, hence it is possible to build an Eulerian path of H having such vertices as endpoints. Note that since H is a multigraph, if an edge e is contained in both T and a path of Π', there are 2 distinct instances of e in H: for the purpose of constructing the Eulerian path, such 2 instances are considered like distinct edges. How to construct an Eulerian path is a well-studied task. However, to allow the conflict resolution performed in Steps 5 and 6 we need an Eulerian path π with a specific structure. In general, there are several occurrences of z and w in an Eulerian path, but we need that the ones which are endpoints of π satisfy proper conditions. More precisely, for any of z and w, we need that if one of its occurrences is endpoint of a path in Π', then such an occurrence is one of the endpoints of π. Note that when z and w are endpoints of the *same* path in Π', only one of such two occurrences can be endpoint of π, so we choose to let the occurrence of z be endpoint of π. In such a case, as well as if w does not occur at all as endpoint of a path in Π', the occurrence of w as endpoint of π is necessarily endpoint of a path p in T. Then we need that any occurrence of w internal to π which is contained in a path p_i in T, is the vertex of p_i having minimum distance in T from z.

To build a path π with the desired properties, we distinguish two cases, according to whether or not there is a path in Π' having both w and z as endpoints.

(1) There exists $q \in \Pi'$, with $q = (u_0, u_1, \ldots, u_{h-1}, u_h)$, $u_0 = z$ and $u_h = w$. Then, both z and w have even degree in T. Observe that in this case, since none of z and w is left exposed by Π, the edge $\{w, y\}$ has not been removed from T during Step 2. We need an Eulerian path π of the form $q, p_1, \ldots, p_l$ with $p_l = (u, \ldots, y, w)$, which can be constructed as follows:
- construct an Eulerian cycle γ on $T \cup (\Pi' \setminus \{q\})$;
- transform γ, without adding any edge, in a path γ' having two occurrences of w as endpoints, by duplicating the occurrence of w adjacent to y, i.e., if x is the other neighbor in γ of that occurrence of w, let $\gamma' = (w, x, \ldots, y, w)$;
- append q to γ' to obtain $\pi = (z, u_1, \ldots, u_{h-1}, w, x, \ldots, y, w)$.

(2) w and z are not endpoints of the same path in Π'. We build an Eulerian path π with the desired properties with the following procedure:

```
Let $B = \Pi'$, $v = z$.
If there exists $q_z \in \Pi'$ with $q_z = (z, \ldots, u_h)$, $u_h \neq w$ then let $B = B \setminus \{q_z\}$ and $v = u_h$.
If there exists $q_w \in \Pi'$ with $q_w = (u, \ldots, w)$, $u \neq z$ then
    Construct an Eulerian path $\gamma$ on $T \cup (B \setminus \{q_w\})$ having $v$ and $u$ as endpoints.
    Append $q_w$ to $\gamma$ obtaining an Eulerian path $\gamma'$ on $T \cup B$ having $v$ and $w$ as endpoints.
else                                                                   (*)
    If $T$ still contains $\{w, y\}$ then
        Construct an Eulerian path $\gamma$ on $(T \cup B) - \{w, y\}$ having $v$ and $y$ as endpoints.
        Append $\{w, y\}$ to $\gamma$ obtaining an Eulerian path $\gamma'$ on $T \cup B$ having $v$ and $w$ as endpoints.
    else construct an Eulerian path $\gamma$ on $T \cup B$ having $v$ and $w$ as endpoints.
If $v \neq z$ then obtain $\pi$ by appending $q_z$ to $\gamma'$
    else let $\pi = \gamma'$.
```

In any of the two cases, π can be considered as an alternating sequence of the form $p_1, q_1, p_2, q_2, \ldots$ or $q_1, p_1, q_2, p_2, \ldots$, where $p_1, p_2, \ldots$ are paths in T and $q_1, q_2, \ldots \in \Pi'$. Note that since T is a tree and π is an Eulerian path, paths $p_1, p_2, \ldots$ are elementary.

In Case 1, π has the form $q, p_1, \ldots, p_l$, where $q \in \Pi'$ has z and w as endpoints, $p_1 = (w, x, \ldots)$, and $p_l = (u, \ldots, y, w)$. This follows from the fact that since w occurs in q, no paths in $\Pi' \setminus \{q\}$ contain w, so any occurrence of w in the cycle γ is internal to a path in T. Then, since $\{w, y\}$ is contained in T and γ is Eulerian, there exists $x \in V$ such that the sequence of vertices x, w, y appears in γ as subpath of a path $\bar{p}$ in T. By duplicating the occurrence of w we divide $\bar{p}$ in two paths p_1, p_l, both contained in T, which become the two terminal subpaths of γ'. We remark that the occurrences of w as endpoint of p_1 and p_l are the only two occurrences of w in π which are endpoint of a path p_i in T. Indeed, if there was a third occurrence of w that is endpoint of a path in T, then that occurrence would also be endpoint of a path q_i in Π', with $q_i \neq q$, which is not possible since Π' is a vertex-disjoint set of paths. Observe also that since edge $\{w, y\}$ is contained in p_l and not in p_1, w is the vertex of p_1 having minimum distance in T from z.

In case 2, if z (resp. w) is an endpoint of a path $q_z = (z, u_1, \ldots)$ (resp. $q_w = (u, \ldots, w)$) in Π', then π has the form $q_z, p_1, \ldots$ (resp. $r, \ldots, p_h, q_w$ with $r \in \{p_1, q_1\}$). This implies that, in this case, there can be no occurrences of w, internal to π, that are endpoint of a path p_i in T. Indeed such an occurrence would also be endpoint of a path $q_j \in \Pi'$, with $|j - i| \leq 1$, but by construction if there is an occurrence of w as endpoint of a path in Π', such an occurrence is not internal to π.

From previous observations on the structure of π, the next remark follows.

Remark 1. *If there exists an occurrence of w internal* *to π, which is at the same time endpoint of a path p in T, then π was constructed according to Case 1 of the procedure and $p = p_1$. Moreover, w is the vertex of p_1 having minimum distance in T from z.*

The following lemma proves some properties of π.

Lemma 3. *Let Π' be the vertex-disjoint path matching obtained at the end of Step 3 and π be the Eulerian path constructed in Step 4. Then:*

- *every vertex $v \in V$ different from w, occurs at most once as endpoint of a path in T;*
- *z occurs as endpoint of either a path in T or a path in Π';*
- *if the occurrence of w which is endpoint of π, is endpoint of a path p_l in T, then each occurrence of w internal to π which is contained in a path p in T, is the vertex of p with the minimum distance in T from z.*

Proof. An internal vertex occurrence is an endpoint of a path p_i in T, if and only if it is also endpoint of a path $q_j \in \Pi'$, with $|j - i| \leq 1$. Since Π' is a set of vertex disjoint paths, a vertex occurs at most once as endpoint of a path in Π', hence for any vertex there can be at most one occurrence internal to π which is endpoint of a path in T. Let $v \in V \setminus \{w, z\}$. Then each occurrence of v is internal, and v occurs at most once as endpoint of a path in T.

Consider now vertex z which, by construction, occurs as endpoint of π. Since Π' is a set of vertex-disjoint paths, z occurs either once or zero times as endpoint of a path $q \in \Pi'$. If it occurs once, in Case 1 as well as in Case 2, π is constructed so that such occurrence is an endpoint of π. Then z can not occur as endpoint of a path in T, since that occurrence should be internal to π, and therefore there should be a second occurrence of z as endpoint of a path in Π', which is not possible. If z does not occur as endpoint of a path in Π', there are also no occurrences of z internal to π which are endpoint of a path in T. But, on the other hand, the occurrence of z as endpoint of π is necessarily endpoint of a path in T.

Recall that, since T is a tree, an occurrence of w which belongs to a path p_i in T is not the vertex of p_i having minimum distance in T from z, if and only if p_i contains the edge $\{w, y\}$. Suppose that the occurrence of w as endpoint of π, is endpoint of a path p_l in T. We analyze separately the two possible cases for the construction of π. In Case 1, by construction, $\{w, y\}$ is contained in p_l which is a terminal subpath of π. In Case 2, w can not occur as endpoint of a path $q_w \in \Pi'$, otherwise by construction q_w would be the terminal subpath of π containing w, instead of p_l. This means that in the procedure constructing π, the case marked as (*) applies, and the edge $\{w, y\}$, if not deleted from T in Step 2, is contained in p_l. Therefore, since π is an Eulerian path, $\{w, y\}$ is not contained in a path in T different from p_l. If an occurrence of w internal to π is contained in a path p_i in T, it is $i \neq l$ because p_l is elementary and the occurrence of w it contains is an endpoint of π. Then p_i does not contain $\{w, y\}$, which implies that the occurrence of w in p_i is the vertex of p_i having the minimum distance in T from z. □

Step 5 The details of the procedure which realizes the main part of Step 5, namely the resolution of some of the conflicts in T, are described in Algorithm 8. Such an algorithm derives from a similar procedure in PMCA, with modifications in order to ensure that there is exactly one occurrence of z in π', and that such an occurrence is indeed an endpoint of π'. In this way, situations like the one illustrated in Figure 1 are not possible, allowing to complete, in Step 6, the conflict resolution process by bypassing at most 4 consecutive edges.

Algorithm 8 is based on the following idea. First, z is picked as root of T. Then, we consider a path p_i in T which, under the orientation with respect to z, 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 minimum spanning tree is divided into several trees, since each bypass building divides a tree into two.

Algorithm 8 Procedure implementing Step 5

```
Input: $T$ and the paths $p_1, p_2, \ldots$ computed in Step 4.
 For each path $p_i = (v_1, \ldots, v_n)$ in $T$ do
    Let $v_j$ be the vertex in $p_i$ of minimum distance in $T$ from $z$.
    If $v_j$ is not an endpoint of $p_i$ then bypass $v_j$.
    Call the resulting path $p'_i$.
Output: The paths $p'_1, p'_2, \ldots$ building a forest $T_f$.
```

Lemma 4. *Consider the path π' obtained at the end of Step 5. The endpoints of π' are w and z. In π', each vertex $v \in V$ occurs either once or twice, and z occurs exactly once.*

Proof. Path π' is built from π substituting each p_i with p'_i. The endpoints of π are w and z. Since Algorithm 8 does not change the endpoints of the paths composing π (neither of those in T, nor of those in Π'), they are also the endpoints of π'.

We first prove some properties of a vertex $v \in V \setminus \{z\}$. Since Π' is a vertex-disjoint path matching, any vertex can occur in π' at most once inside paths in Π'. Moreover, assume that there are two distinct occurrences of $v \in V \setminus \{z\}$ internal to paths p'_i and p'_j in T_f. 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 vertices v_1, v_2, v_3, v_4 is closer to z than v. Thus, v is bypassed from at least one of the paths p_i, p_j during Algorithm 8, because v is closer to z than all other vertices of that path. This is a contradiction to our assumption, hence for any vertex $v \in V \setminus \{z\}$, there is at most one occurrence of v internal to a path in T_f.

Consider $v \in V \setminus \{w, z\}$. Since T is a tree, there is a neighbor v_1 of v such that the distance in T of v_1 from z is less than the distance in T of v from z. Since $v \neq w$, then $\{v, v_1\} \neq \{w, y\}$, so $\{v, v_1\}$ is surely not removed from T in Step 3. This implies that $\{v, v_1\}$ is contained in exactly one of the paths $p_1, p_2, \ldots$ in T, say p_i, because $p_1, p_2, \ldots$ are part of the Eulerian path π. Then, since p_i contains v_1, v is not the vertex of p_i having minimum distance in T from z, hence the occurrence of v in p_i is not bypassed during Step 5 and there is at least one occurrence of v in π'.

On the other hand, since v is not an endpoint of π', if there is an occurrence of v as endpoint of a path p'_i in T_f, then such an occurrence is also endpoint of a path in Π'. Since there can be at most one occurrence of v inside paths in Π', any other occurrence of v has to be internal to some path p'_i in T_f. Therefore, a vertex $v \in V \setminus \{w, z\}$ can have at most one occurrence in a path in Π' (possibly endpoint of a path in T_f, too) and at most one occurrence internal to a path in T_f, for a total of at most two occurrences in π'.

Consider now w. If the occurrence of w as endpoint of π' is endpoint of a path $q_w \in \Pi'$, then any other occurrence of w is internal to some path p' in T_f. As shown above for a generic vertex in $V \setminus \{z\}$, there is at most one occurrence of w internal to a path in T_f, so in total there are at most two occurrences of w in π'.

On the other hand, if the occurrence of w as endpoint of π' is endpoint of a path p'_l in T_f, then, after Step 4, the occurrence of w as endpoint of π is endpoint of a path p_l in T, and Algorithm 8 transforms p_l in p'_l. Any occurrence of w internal to π is either internal to a path p_i in T, with $i \neq l$, or is contained in a path $q \in \Pi'$. By Lemma 3, any occurrence of w internal to a path p_i in T is the vertex of p_i having the minimum distance in T from z. Hence any occurrence of w internal to a path p_i in T is bypassed during the run of Algorithm 8, and does not occur in π'. Since Π' is a vertex disjoint set of paths, there can be at most one occurrence of w inside a path in Π'. Therefore, there is at most one occurrence of w internal to π', so, again, there are at most two occurrences of w in π'.

Since z is the vertex of minimum distance from itself, any occurrence of z internal to a path in T is bypassed during the run of Algorithm 8. Moreover, by Lemma 2, there are no occurrences of z internal to a path in Π'. By Lemma 3, z occurs exactly once as endpoint of a path, either of one in T or of one in Π'. This one is the unique occurrence of z in π', since there are no occurrences of z internal to any path. □

From Algorithm 8, we obtain immediately the next observation which will be used in the following.

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

Step 6 Before describing how to realize Step 6, we state the following lemmas, which will be used to prove that bypasses, at the end of the whole algorithm, have size at most 4.

Lemma 5. *Let p_h be one of the paths in T composing the Eulerian path π, and let p'_h be the path constructed from p_h by Algorithm 8. Let $v \in V \setminus \{w\}$ be an endpoint of p_h (and of p'_h). If v is not the vertex of p_h having minimum distance in T from z, then v is not a conflict in π'.*

Proof. Suppose v is not the vertex of p_h having minimum distance in T from z, which immediately implies $v \neq z$. Then $v \in V \setminus \{z, w\}$, and the occurrence of v in π as endpoint of p_h is also endpoint of a path $q_v \in \Pi'$. Since Π' is a vertex disjoint set of paths, there are no other occurrences of v in π inside paths from Π'.

Suppose there is an occurrence of v in π inside a path p_i in T, with $i \neq h$. By Lemma 3, the occurrence of v in p_i is internal to p_i. Let u be the unique neighbor of v in T such that u has distance in T from z less than that of v. Since v is not the vertex of p_h having minimum distance in T from z, the edge $\{v, u\}$ is contained in p_h. This means that $\{v, u\}$ is not contained in paths in T other than p_h, because π is Eulerian. It follows that v is the vertex of p_i having minimum distance in T from z. Then the occurrence of v in p_i is at the same time internal to p_i and the vertex of p_i having the minimum distance in T from z. Hence v is bypassed by Algorithm 8 in all paths other than p'_h, so it is not a conflict in π'. □

The crucial property which gives the desired bound on the size of bypasses, is stated in the following lemma. A similar result is proved in [4], but here a different proof is needed.

Lemma 6. *In the path π', between each two bypasses there is at least one vertex that is not a conflict.*

Proof. Note that "between" includes the case that the claimed vertex may be endpoint of one or both edges used as bypass. Let p_i be one of the paths in T composing the Eulerian path π and let p'_i be the path constructed from p_i by Algorithm 8. Then p_i and p'_i have the same endpoints. An important observation is that since T is a tree, at least one of the endpoints of p_i is not the vertex of p_i having the minimum distance in T from z, and if Algorithm 8 constructs a bypass in p'_i then both the endpoints of p_i are not the vertices of p_i having the minimum distance in T from z. By Remark 1, we have that if an endpoint v of p_i is at the same time internal to π and not the vertex of p_i having the minimum distance in T from z, it is $v \neq w$. Then we conclude, by Lemma 5, that v is not a conflict in π'.

We say that two bypasses in π' are *close*, if there are no other bypasses between them on π'. To show the thesis, it is enough to consider two bypasses which are close and to prove that, in π', there is at least one vertex that is not a conflict between them.

Suppose that at least one of the two considered bypasses was constructed by Algorithm 8 on a path p_i in T, producing $p'_i \in T_f$, and let v be the endpoint of p_i (and p'_i) between the considered bypasses. Since p'_i contains a bypass, both of the endpoints of p_i are not the vertices of p_i having the minimum distance in T from z. But v is also internal to π, so we conclude, as in the above observation, that v is not a conflict in π'.

On the other hand, suppose that both bypasses lie on paths from Π', and let p_i be a path in T which is between the two bypasses. At least one of the endpoints of p_i, say v, is not the vertex of p_i having the minimum distance in T from z, and clearly v is internal to π. Then, again as in the above observation, we have that v is not a conflict in π'. □

We are now ready to describe the procedure which realizes Step 6. It derives from a similar procedure in algorithm PMCA, with modifications to avoid that more than two consecutive vertices of π' are bypassed. To this aim, Algorithm 9, immediately after bypassing a vertex v, resolves, as *not* bypassed, an unresolved conflict in π' adjacent to v, if any.

Lemma 7. *Algorithm 9 terminates after resolving all conflicts in π', and it generates bypasses of size at most 4 overall, i.e., taking into account that some edges of the input path π' may be bypasses of size 2 themselves. The endpoints of the returned Hamiltonian path π'' are w and z.*

Algorithm 9 Procedure implementing Step 6

```
Input: A path $\pi'$ on $G$ where $w$ and $z$ are the endpoints, $z$ occurs once, and all the other vertices of $V$ occur once or twice.
 If $w$ is a conflict in $\pi'$ then let $u$ be the occurrence of $w$ which is not endpoint of $\pi'$ else let $u$ be an arbitrary occurrence of a conflict in $\pi'$.
 Bypass $u$ (with a bypass of size 2).
 While there are conflicts remaining in $\pi'$ do
   If occurrence $u$ has at least one occurrence of an unresolved conflict in $\pi'$ as neighbor
   then
     Choose $v$ between the neighbors of $u$ which are unresolved conflict in $\pi'$ so that:
       If there is a bypassed vertex occurrence $t$ such that each vertex occurrence internal to the elementary subpath $p$ in $\pi'$ connecting $u$ and $t$, is an occurrence of an unresolved conflict in $\pi'$
       then let $v$ be the neighbor of $u$ that belongs to $p$
       else let $v$ be an arbitrary neighbor of $u$ which is an unresolved conflict in $\pi'$.
     Bypass the other occurrence of $v$ in $\pi'$ (the one not chosen by previous statement) so letting $v$ be a resolved, not bypassed conflict in $\pi'$.
   else bypass an arbitrary occurrence of a conflict in $\pi'$.
   Let $u$ be the vertex occurrence bypassed in the previous statement.
Output: A Hamiltonian path $\pi''$ of $G$ having $w$ and $z$ as endpoints.
```

Proof. The only difference between Algorithm 9 and Procedure 3 of [4] is that in the latter, the first conflict in π' which is resolved (outside the main loop), is an arbitrary one. Hence the same reasoning as in the proof of Claim 6 of [4] shows the first part of our thesis, too. Please note that such a reasoning uses the result of Lemma 6, which in our case needs a proof significantly different from the one given in [4].

By Lemma 4, z and w are the endpoints of π', and z is not a conflict in π'. This latter fact implies that the occurrence of z is left unaltered by Algorithm 9. If w is a conflict in π', Algorithm 9 resolves it immediately, by bypassing the occurrence of w which is not an endpoint. So z and w are the endpoints of π''. ☐

The following lemma, analyzes the quality of the approximation provided by Algorithm 5.

Lemma 8. *The cost of the Hamiltonian path π'' returned by Algorithm 5 is less than $\frac{3}{2}\beta^2$ the cost of P^*, at most $\frac{3}{2}\beta^2$ the cost of P^*_s, and at most $\frac{5}{3}\beta^2$ the cost of P^*_{st}.*

Proof. Since any spanning path of G is a tree, $c(T) \leq c(P^*) \leq c(P^*_s) \leq c(P^*_{st})$. In [8] it is shown that if G is a metric graph, the cost of a minimum matching for U which leaves $2-k$ vertices exposed, is less than $\frac{1}{2}c(P^*)$ when $k=0$, it is no more than $\frac{1}{2}c(P^*_s)$ when $k=1$, and it is no more than $\frac{2}{3}c(P^*_{st})$ when $k=2$. Those proofs are based on the fact that in a metric graph, the cost of an edge $\{u,v\}$ is a lower bound for the cost of any path having u and v as endpoints. Then the same arguments, with the only change of using shortest paths forming Π instead of the direct edges forming a minimum matching, show that in our case it is $c(\Pi) < \frac{1}{2}c(P^*)$ when $k=0$, $c(\Pi) \leq \frac{1}{2}c(P^*_s)$ when $k=1$, and $c(\Pi) \leq \frac{2}{3}c(P^*_{st})$ when $k=2$. From Lemma 7 we have that in the path π'' returned by Algorithm 5 at most 4 consecutive edges of $T \cup \Pi$ are bypassed with a new single edge (taking into account the combined effects of Steps 3-6). This may increase the cost of π'' by a factor of at most β^2 with respect to the cost of $T \cup \Pi$. Consequently, we have $c(\pi'') < \frac{3}{2}\beta^2 c(P^*)$, $c(\pi'') \leq \frac{3}{2}\beta^2 c(P^*_s)$ and $c(\pi'') \leq \frac{5}{3}\beta^2 c(P^*_{st})$. ☐

The following theorem summarizes the results of this section:

Theorem 3. *For every $\beta > 1$, there are $(\frac{3}{2}\beta^2)$-approximation algorithms for Δ_β-HPP$_0$ and Δ_β-HPP$_1$, and a $(\frac{5}{3}\beta^2)$-approximation algorithm for Δ_β-HPP$_2$. The algorithms run in $O(n^3)$ time.*

Proof. Algorithms PMCA-HPP$_k$, with $k=0,1,2$, have the properties required by the thesis. The correctness of the algorithms is proved in Lemmas 1, 2, 3, 4, 5, 6, 7. The claimed approximation ratios are proved in Lemma 8. The upper bound for the time complexity is proved observing that in Step 2 the minimum path matching is computed in $O(n^3)$ time, while the remaining steps can be implemented in $O(n^2)$ time. ☐

4. Conclusions

In this work, we successfully extended the class of graphs for which HPP is approximable (with constant approximation ratio, in polynomial time). Please note that for every graph G, it is possible to find a suitable value of β so that G satisfies the β-triangle inequality. Hence HPP is approximable for every input graph, although as β grows the approximation ratio quickly becomes large.

The HPP$_0$ and the HPP$_1$ are similar to the TSP, and we expect the three problems to exhibit the same behavior with respect to approximability. Indeed, in the metric case, the best known approximation ratios are the same for all three problems [8]. In this paper we proved that the same happens also when the input graph satisfies the β-triangle inequality for $1 \leq \beta \leq 3$. To extend this result to other (possibly any) values of β, further studies are required, especially concerning the possibility to carry over to HPP the approach used in [3].

The HPP$_2$, instead, seems to have slightly different characteristics from an approximability perspective. Indeed, already in the metric case, the best known approximation ratio for the HPP$_2$ is higher than the one known for the TSP. In this work, we obtained an approximation ratio for Δ_β-HPP$_2$ which naturally generalizes the one known for the metric case.

References

[1] Thomas Andreae. On the traveling salesman problem restricted to inputs satisfying a relaxed triangle inequality. *Networks: An International Journal*, 38(2):59–67, 2001.

[2] Thomas Andreae and Hans-Jürgen Bandelt. Performance guarantees for approximation algorithms depending on parametrized triangle inequalities. *SIAM Journal on Discrete Mathematics*, 8(1):1–16, February 1995.

[3] Michael Bender and Chandra Chekuri. Performance guarantees for the TSP with a parameterized triangle inequality. In Frank K. H. A. Dehne, Arvind Gupta, Jörg-Rüdiger Sack, and Roberto Tamassia, editors, *Algorithms and Data Structures, 6th International Workshop, WADS '99, Vancouver, British Columbia, Canada, August 11-14, 1999, Proceedings*, volume 1663 of *Lecture Notes in Computer Science*, pages 80–85. Springer, August 1999.

[4] Hans-Joachim Böckenhauer, Juraj Hromkovič, Ralf Klasing, Sebastian Seibert, and Walter Unger. Towards the notion of stability of approximation for hard optimization tasks and the traveling salesman problem. *Theoretical Computer Science*, 285(1):3–24, July 2002.

[5] Nicos Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. Technical report, Graduate School of Industrial Administration, Carnegy–Mellon University, 1976.

[6] Herbert Fleischner. On spanning subgraphs of a connected bridgeless graph and their application to dt graphs. *Journal of Combinatorial Theory*, 16:17–28, 1974.

[7] Herbert Fleischner. The square of every two-connected graph is hamiltonian. *Journal of Combinatorial Theory*, 16:29–34, 1974.

[8] Han J. A. Hoogeveen. Analysis of christofides' heuristic: Some paths are more difficult than cycles. *Operational Research Letters*, 10:291–295, 1991.

[9] Hang Tong Lau. *Finding a Hamiltonian cycle in the square of a block*. PhD thesis, McGill University, February 1980.

[10] Hang Tong Lau. Finding eps-graphs. *Monatshefte für Math.*, 92:37–40, 1981.

Received 22 Dec 2004; revised 25 April 2005; accepted 10 July 2005